YUIMA: An **R** Framework for Simulation and Inference for Stochastic Processes

Stefano M. Iacus

University of Milan
VOICES from the Blogs & AlgoFinance
R Foundation for Statistical Computing
YUIMA Core Team
Big Data Committee, Italian National Institute of Statistics

QFin@Work - Rome, 4 May 2018

Overview of the `yuima` package

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

The `yuima` package is developed by academics 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!

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.

The main classes of stochastic processes, all multidimensional and eventually parametric models, are:

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.

The main classes of stochastic processes, all multidimensional and eventually parametric models, are:

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

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.

The main classes of stochastic processes, all multidimensional and eventually parametric models, are:

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

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.

The main classes of stochastic processes, all multidimensional and eventually parametric models, are:

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

$$M_t = m_0 + \sum_{i=1}^{N_t} Y_{\tau_i}, \quad N_t \sim \text{Poisson}(\Lambda(t, \theta)), \quad Y_{\tau_i} \ i.i.d. \sim \ \mathcal{L}(\theta)$$

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.

The main classes of stochastic processes, all multidimensional and eventually parametric models, are:

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

$$M_t = m_0 + \sum_{i=1}^{N_t} Y_{\tau_i}, \quad N_t \sim \text{Poisson}(\Lambda(t,\theta)), \quad Y_{\tau_i}\ i.i.d. \sim\ \mathcal{L}(\theta)$$

- Diffusions with jumps or pure Lévy processes

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

where Z_t is a Lévy process.

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.

The main classes of stochastic processes, all multidimensional and eventually parametric models, are:

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

$$M_t = m_0 + \sum_{i=1}^{N_t} Y_{\tau_i}, \quad N_t \sim \text{Poisson}(\Lambda(t, \theta)), \quad Y_{\tau_i}\ i.i.d. \sim\ \mathcal{L}(\theta)$$

- Diffusions with jumps or pure Lévy processes

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

  where Z_t is a Lévy process.
- CARMA(p, q) & COGARCH(p, q) models driven by Lévy noise

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.

The main classes of stochastic processes, all multidimensional and eventually parametric models, are:

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

$$M_t = m_0 + \sum_{i=1}^{N_t} Y_{\tau_i}, \quad N_t \sim \text{Poisson}(\Lambda(t, \theta)), \quad Y_{\tau_i} \ i.i.d. \sim \ \mathcal{L}(\theta)$$

- Diffusions with jumps or pure Lévy processes

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

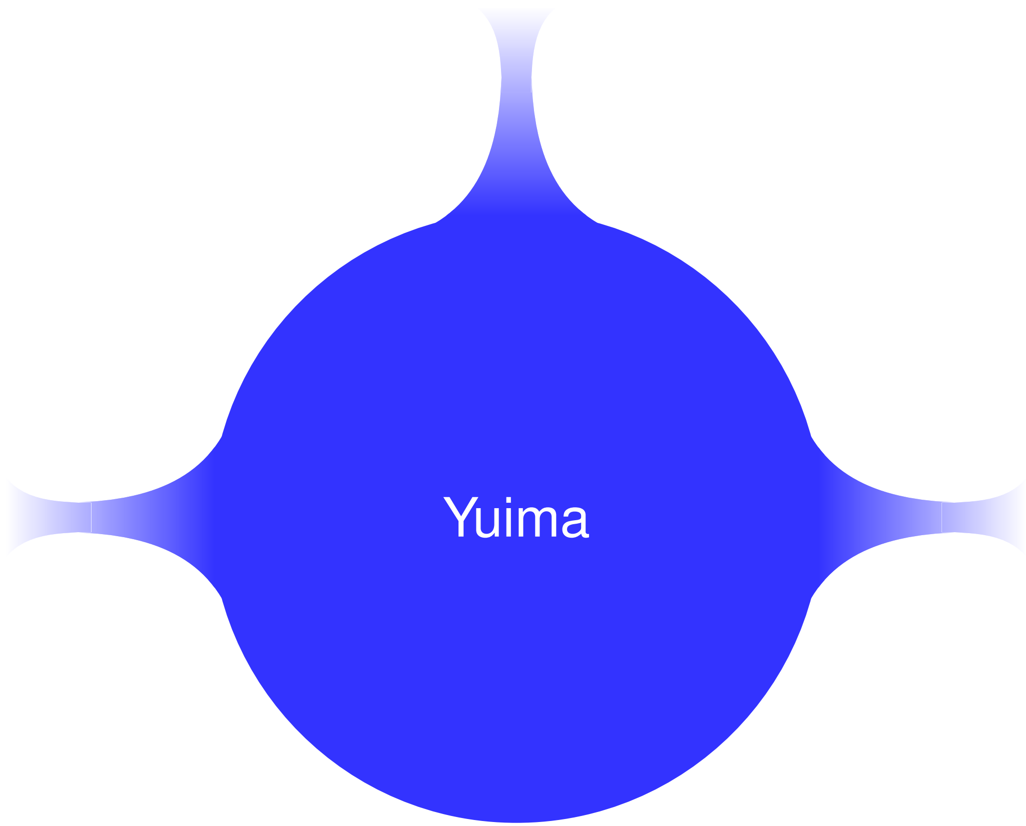
  where Z_t is a Lévy process.
- CARMA(p, q) & COGARCH(p, q) models driven by Lévy noise
- Point Processes (including Hawkes processes)

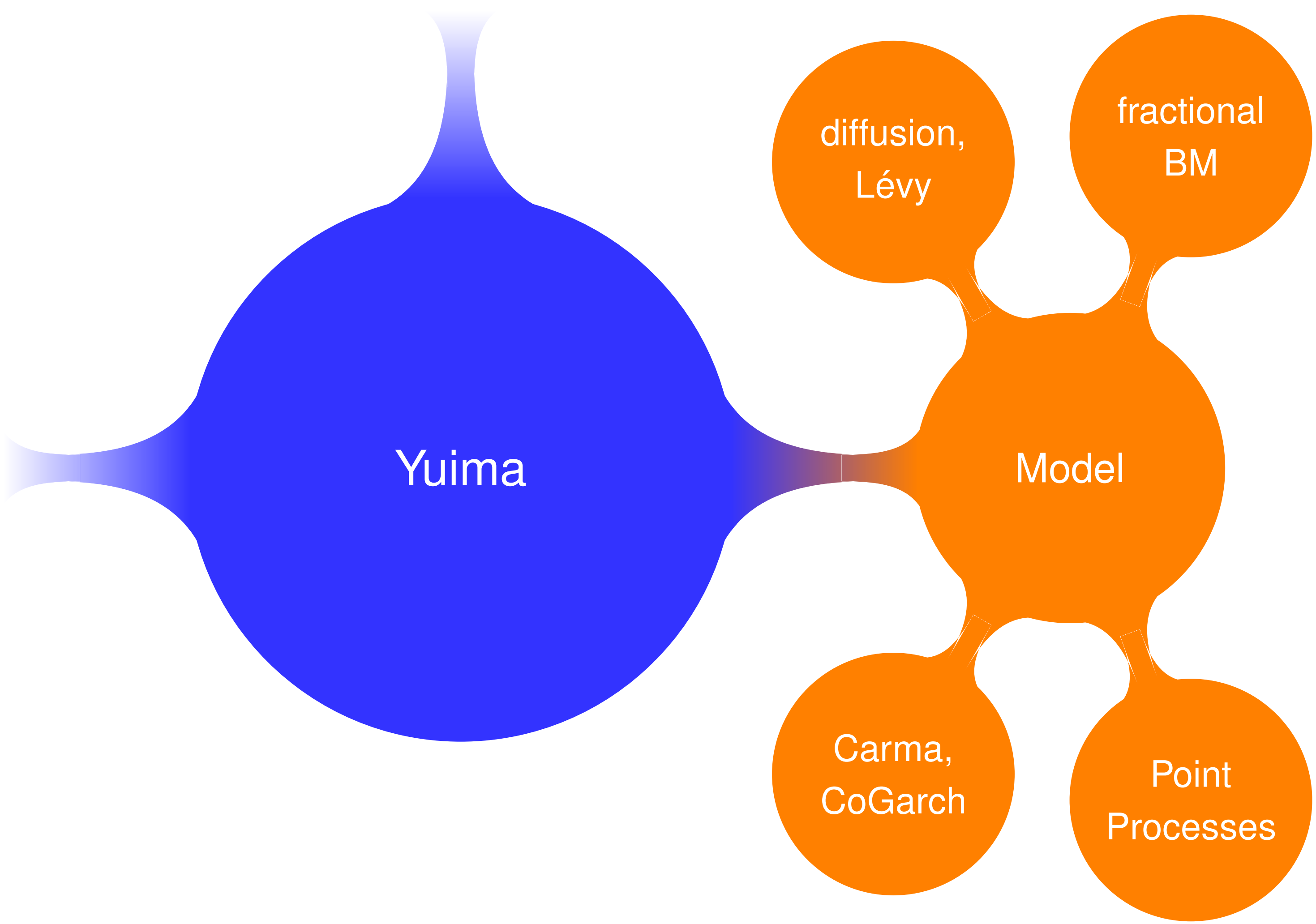
The main object is the yuima object which allows to describe the model in a mathematically sound way.
An object of type yuima contains the following slots:

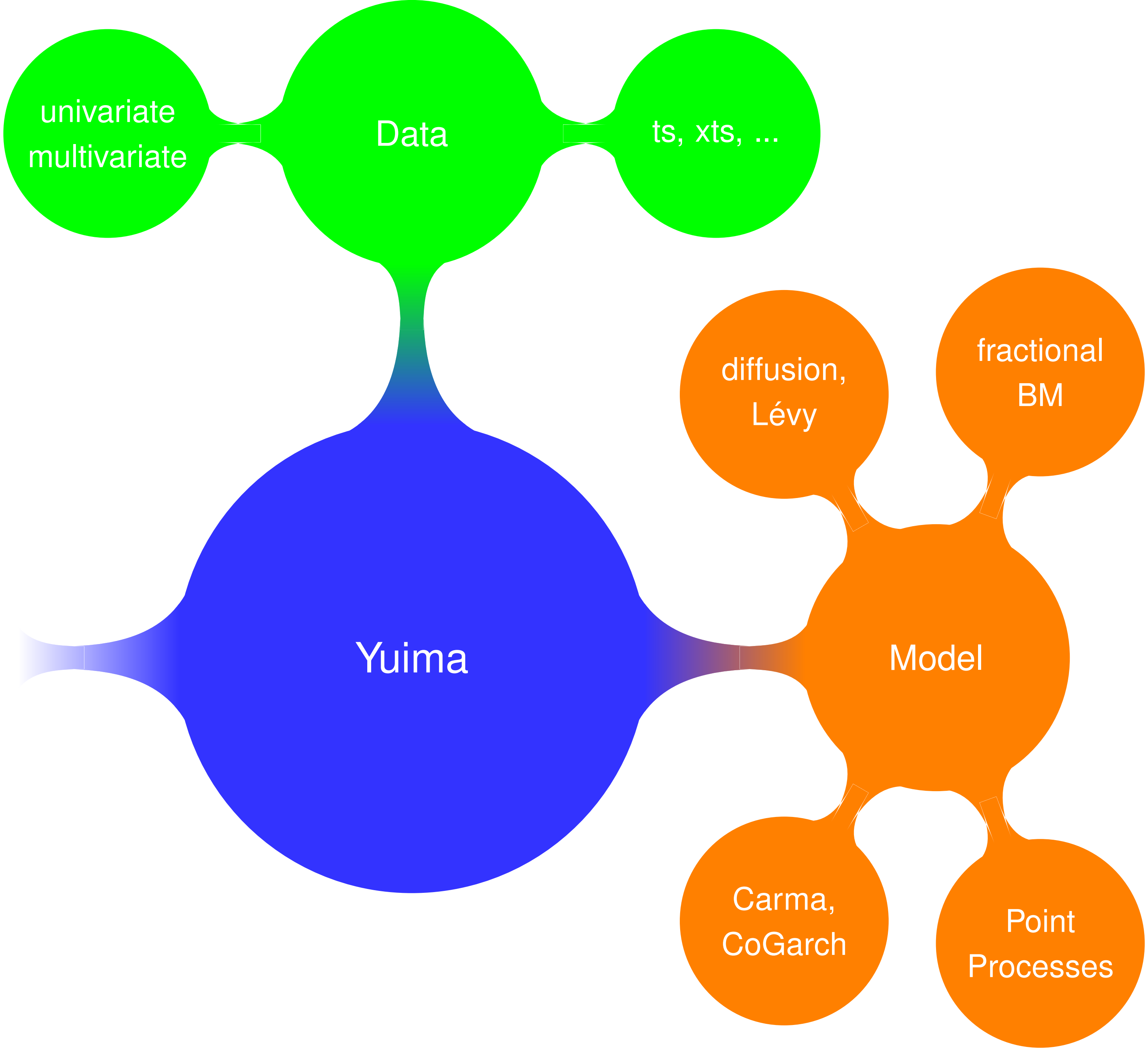
- data: an object of class yuima.data that stores real or simulated data.
- model: an object of class yuima.model that is a mathematical description of the model.
- sampling: an object of class yuima.sampling that is the sampling structure.
- characteristic: additional informations about model such as number of equations and time-scale.
- functional: This slot is filled if we want to compute the functional of stochastic process.

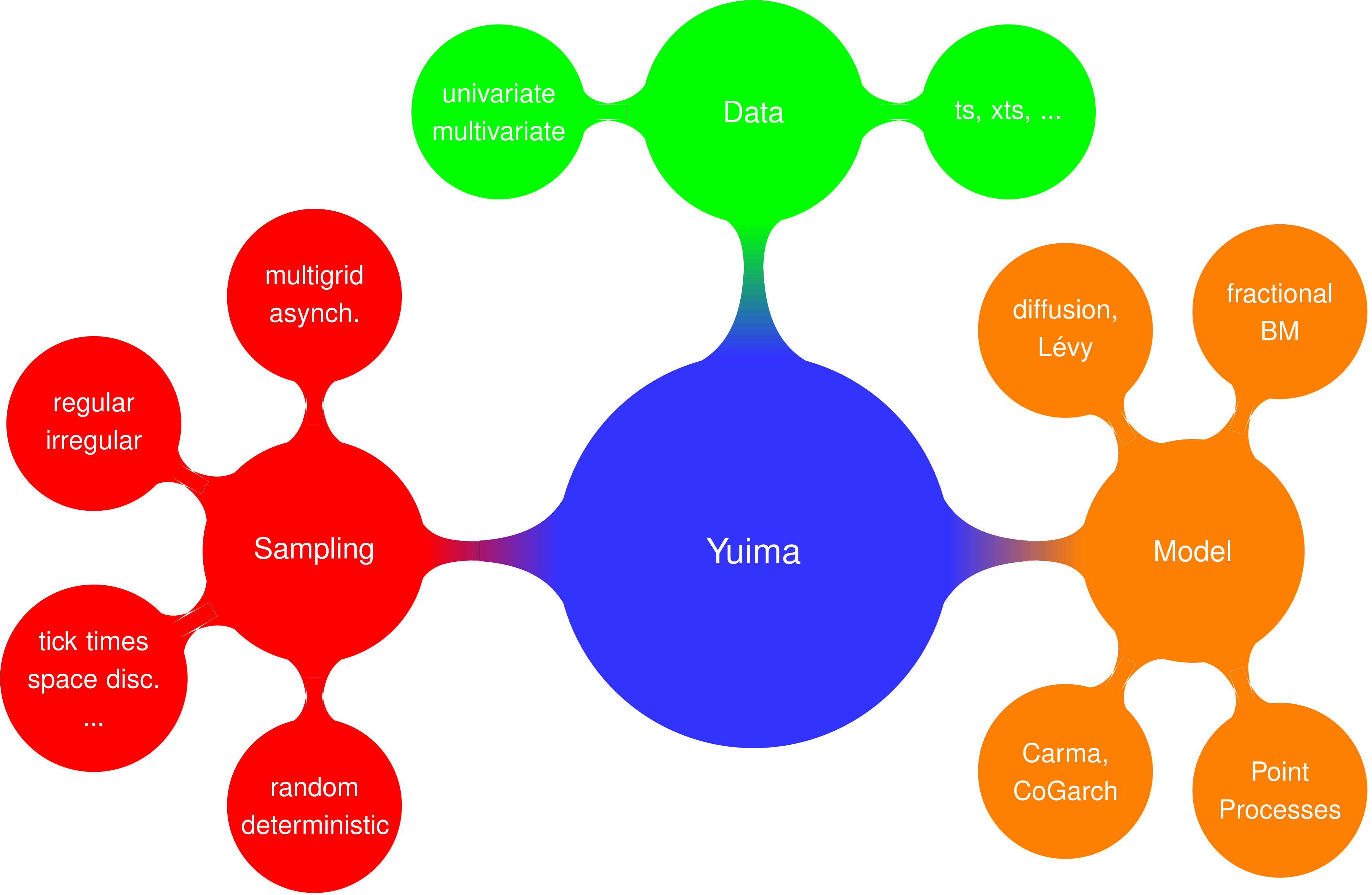
The package exposes very few generic functions like simulate, qmle, plot, etc. and some other model constructor functions, such as, setModel, setCarma, setCogarch, setPpr, setPoisson, setFunctional , setMap, setIntegral, setLaw and other special functions for specific inference tasks.

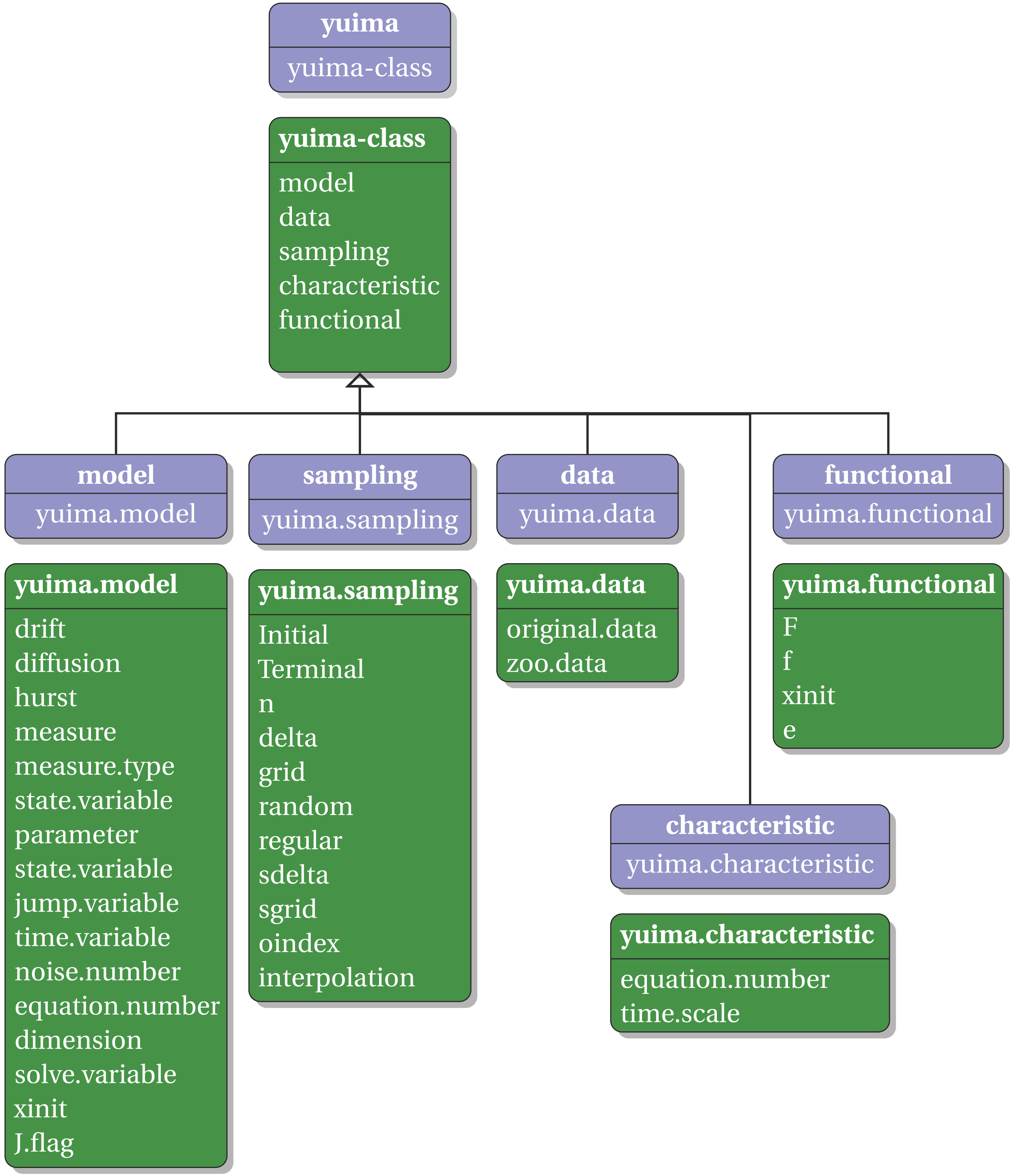
What contains a yuima object ?


Yuima

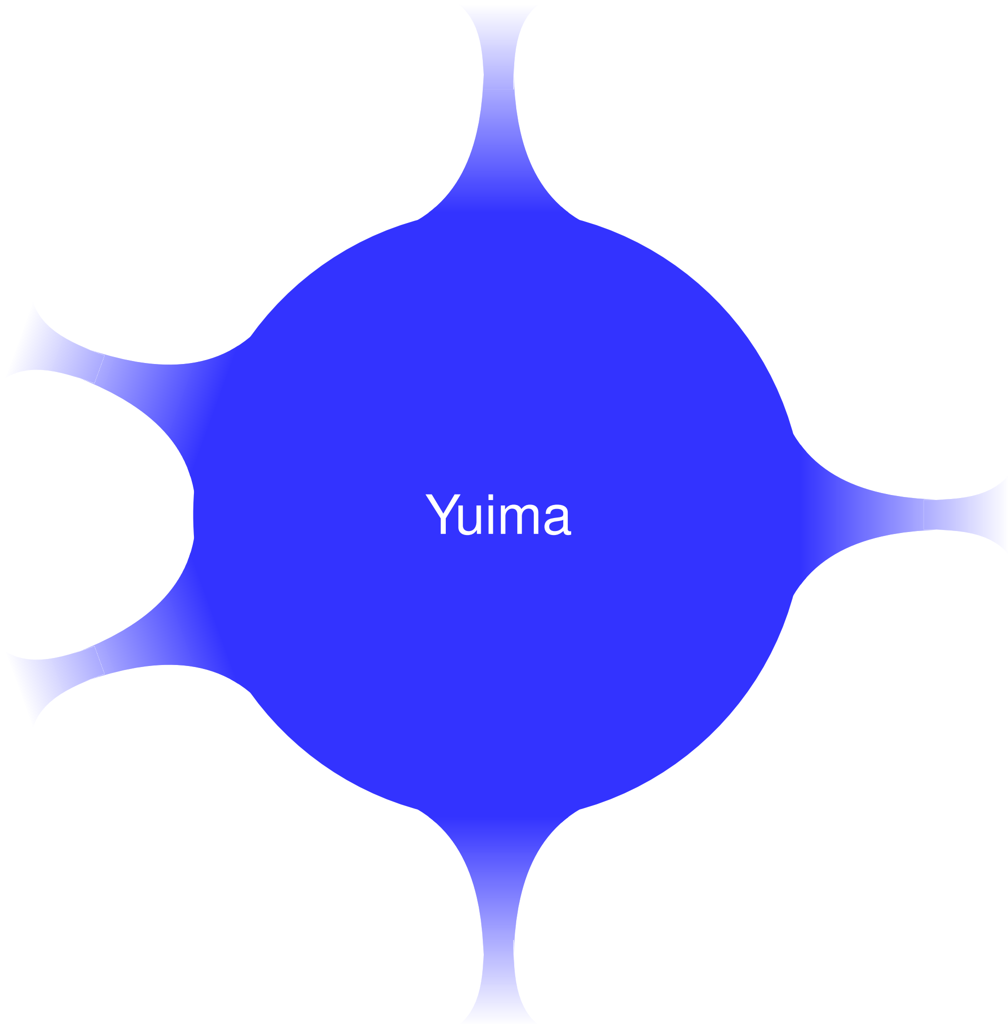

Yuima
Model
diffusion, Lévy
fractional BM
Carma, CoGarch
Point Processes

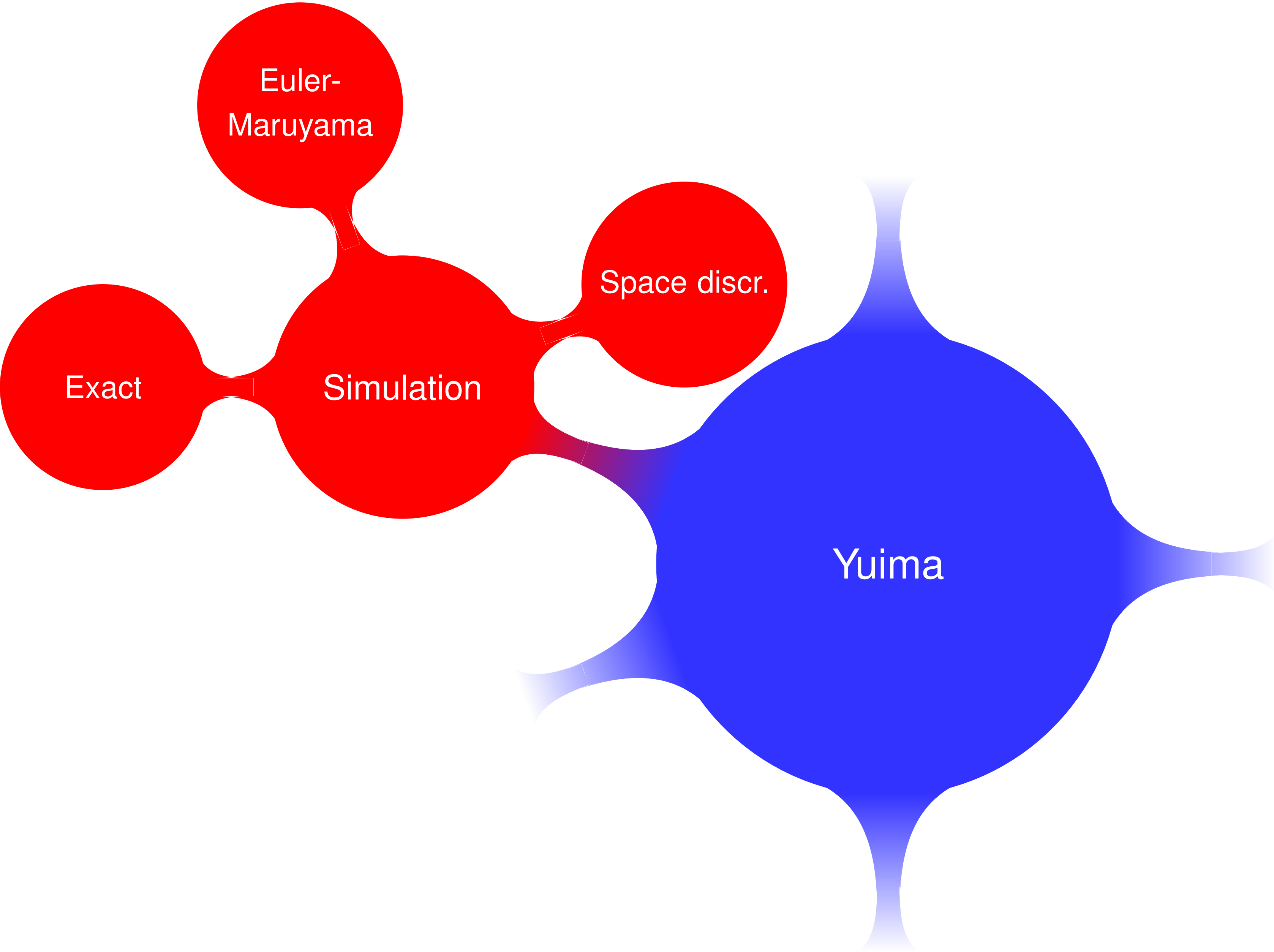

univariate
multivariate
Data
ts, xts, ...
diffusion,
Lévy
fractional
BM
Yuima
Model
Carma,
CoGarch
Point
Processes

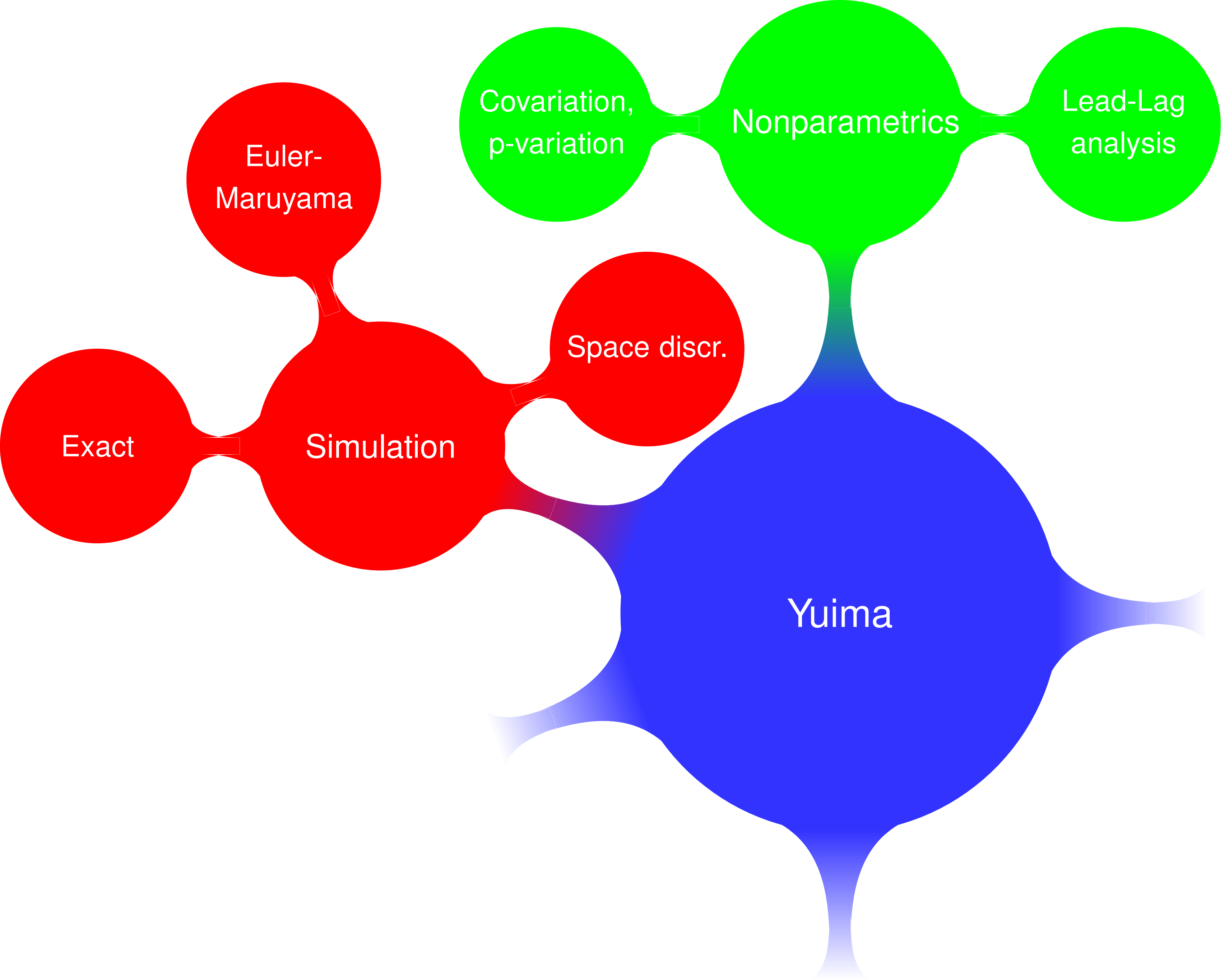

univariate
multivariate
Data
ts, xts, ...
multigrid
asynch.
diffusion,
Lévy
fractional
BM
regular
irregular
Sampling
Yuima
Model
tick times
space disc.
...
random
deterministic
Carma,
CoGarch
Point
Processes

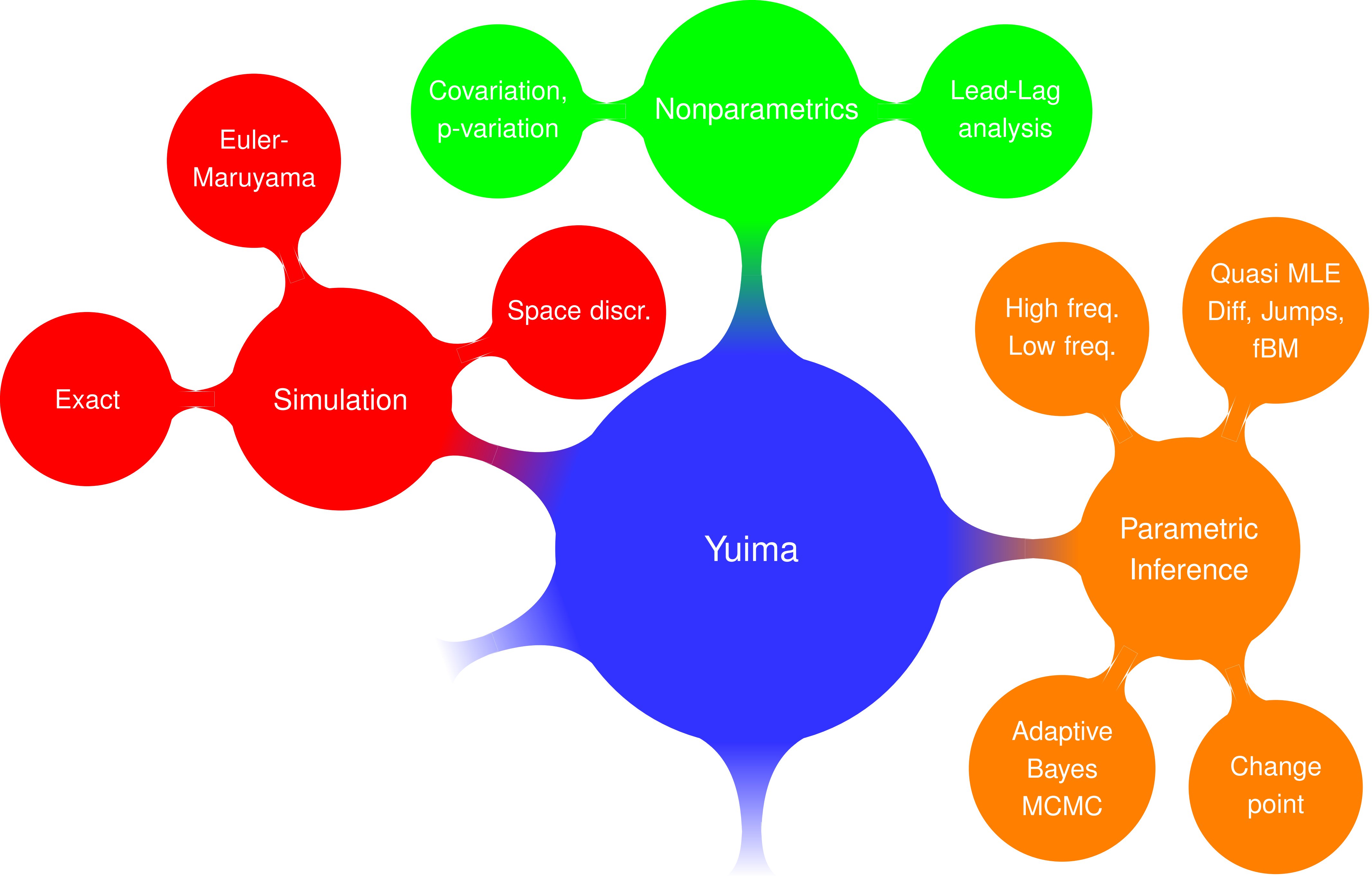

yuima
yuima-class
yuima-class
model
data
sampling
characteristic
functional
model
yuima.model
sampling
yuima.sampling
data
yuima.data
functional
yuima.functional
yuima.model
drift
diffusion
hurst
measure
measure.type
state.variable
parameter
state.variable
jump.variable
time.variable
noise.number
equation.number
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

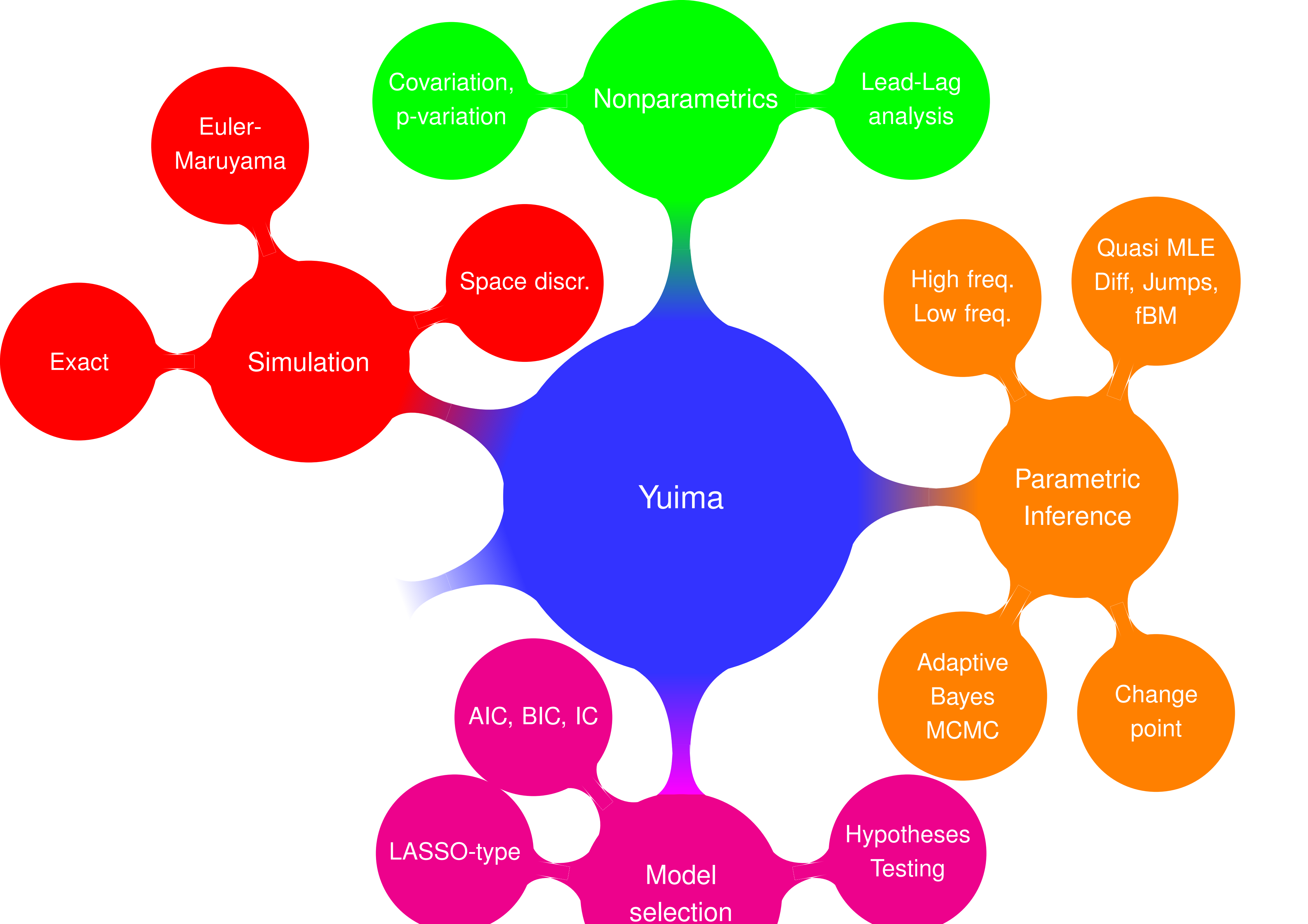
What is possible to do with a yuima object in hands?


Yuima


Euler-
Maruyama
Space discr.
Exact
Simulation
Yuima


Covariation,
p-variation
Nonparametrics
Lead-Lag
analysis
Euler-
Maruyama
Space discr.
Exact
Simulation
Yuima


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


Yuima
Nonparametrics
Covariation, p-variation
Lead-Lag analysis
Simulation
Euler-Maruyama
Space discr.
Exact
Parametric Inference
High freq. Low freq.
Quasi MLE Diff, Jumps, fBM
Adaptive Bayes MCMC
Change point
Model selection
AIC, BIC, IC
LASSO-type
Hypotheses Testing


Yuima
Nonparametrics
Covariation, p-variation
Lead-Lag analysis
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
AIC, BIC, IC
LASSO-type
Hypotheses Testing

Which tools have been developed so far?

- Quasi-MLE for multidimensional diffusions (Yoshida, 1992, 2005).
- Quasi-MLE for SDE with jumps of Poisson type (Shimizu & Yoshida, 2006)
- MLE for inhomogeneous Compound Poisson processes (Kutoyants, 1998)
- Adaptive Bayes type estimators for diffusion processes (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)
- Simple AIC and LASSO-type model selection (De Gregorio & Iacus, 2010)
- Hypotheses testing (De Gregorio & Iacus, 2012)
- Asynchronous covariance estimator of Yoshida-Hayashi (2005) for multidimensional Itô processes
- Estimation for the fractional OU process (Brouste & Iacus, 2013)
- Lead-Lag estimation (Hoffman, Rosenbaum & Yoshida, 2013)
- Quasi-MLE for CARMA(p,q) models with Lévy innovations (Iacus & Mercuri, 2014)
- GMM and Quasi-MLE for COGARCH(p,q) models with Lévy innovations (Iacus, Mercuri & Rroji, 2016, 2018)
- Estimation for general Point Processes (Mercuri & Yoshida, 2016), Hawkes processes with applications to LBO (Limit Book Order)
- a dedicated GUI to exploit graphically some of the above functionalities

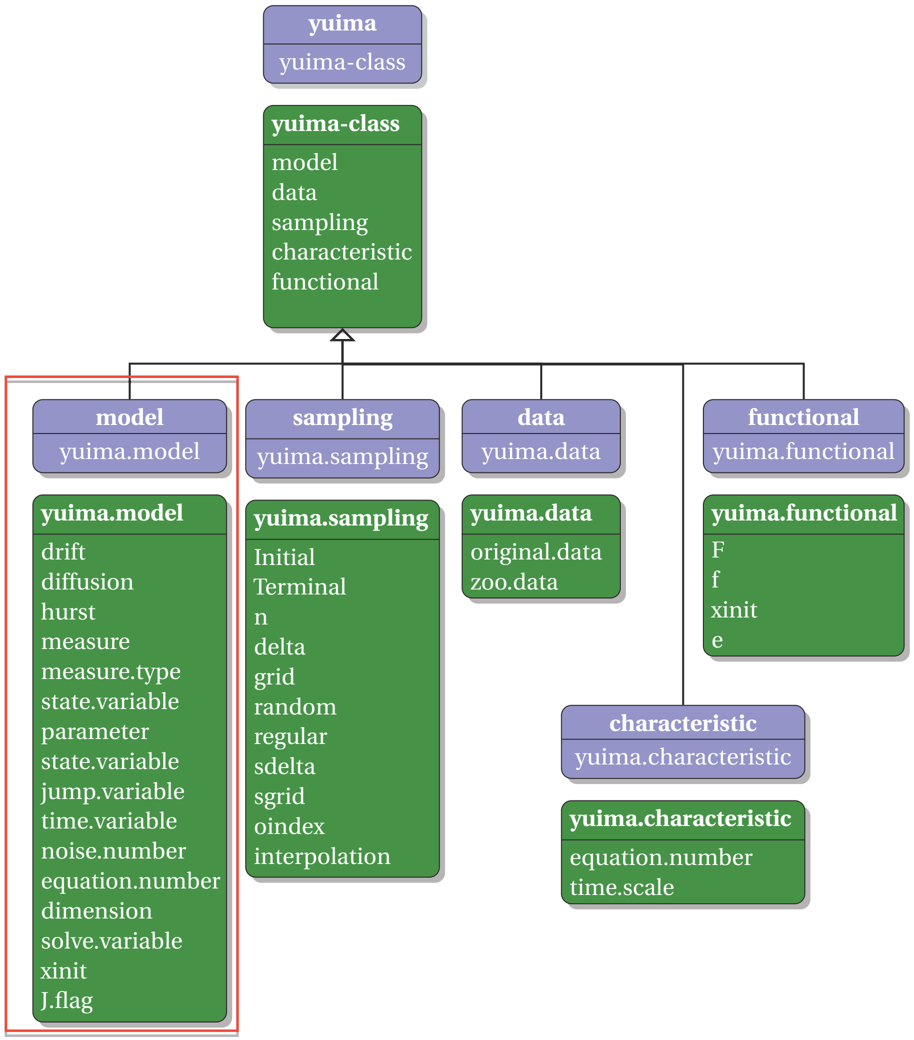
Just not to be too vague, let us consider the exact formulations of some of the problems which can be handled by the `yuima` package.

How does it work?

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


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

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

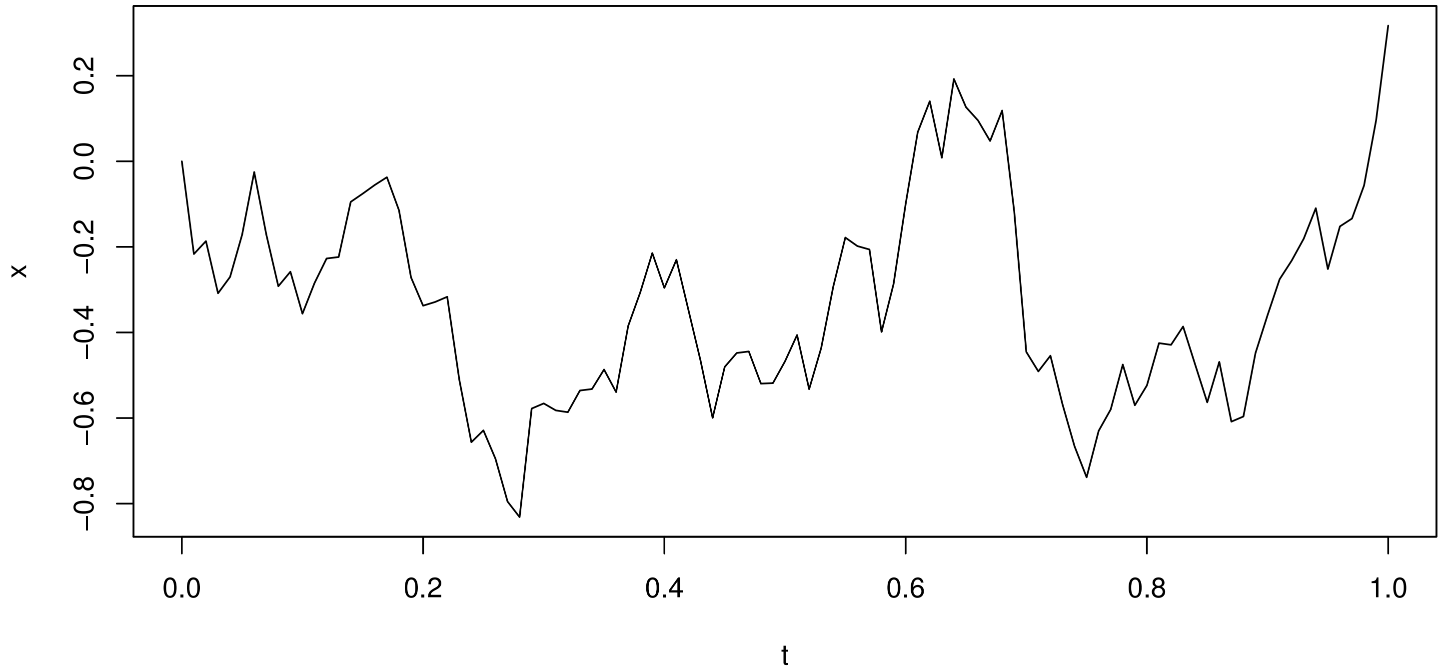


```
> mod1 <- setModel(drift = "-3*x", diffusion = "1/(1+x^2)")
```

```
> 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

```
> mod1 <- setModel(drift = "-3*x", diffusion = "1/(1+x^2)")
> set.seed(123)
> X <- simulate(mod1)
> plot(X)
```

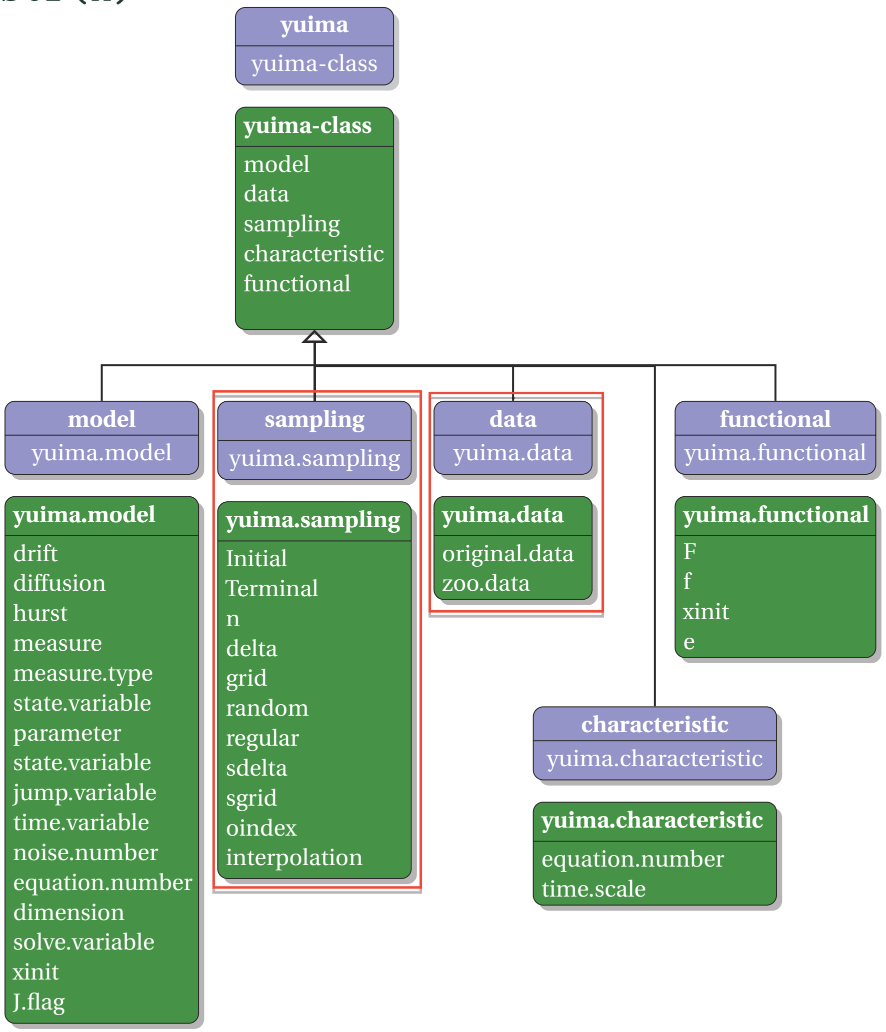


The simulate function fills the slots data and sampling

```
> 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"
```

The simulate function fills the slots data and sampling

```
> 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
```

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

```
> 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
```

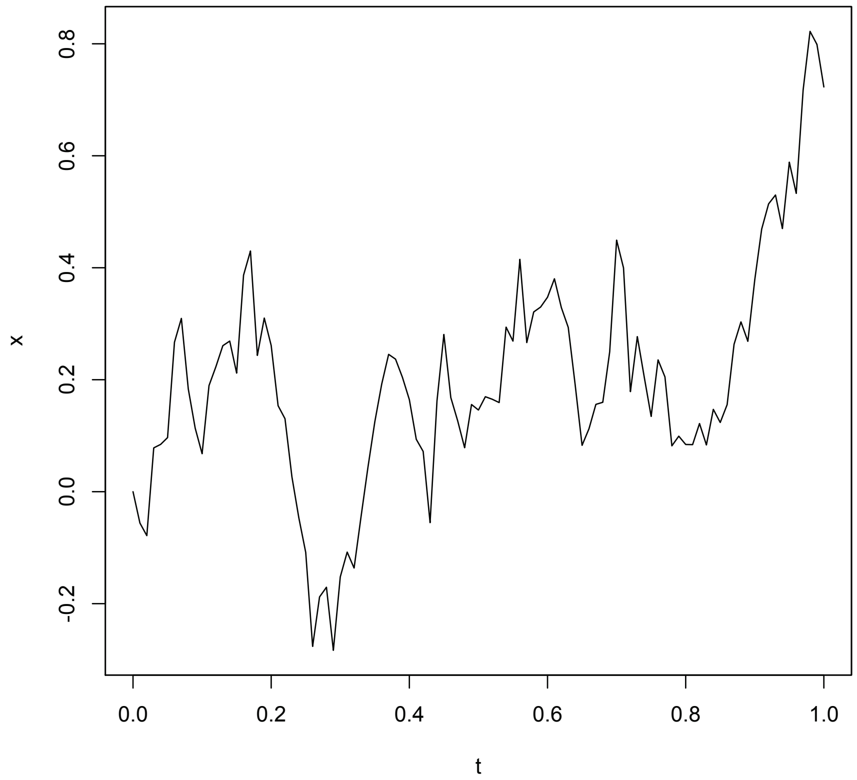
```
> mod2
```

Automatic extraction of the `parameters` for further inference

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

And this can be simulated specifying the parameters

```
> set.seed(123)
> plot(simulate(mod2, true.par=list(theta=1, gamma=3)))
```



$$dX_t^1 = -3X_t^1 dt + dW_t^1 + X_t^2 dW_t^3$$
$$dX_t^2 = -(X_t^1 + 2X_t^2) dt + X_t^1 dW_t^1 + 3 dW_t^2$$

has to be organized into matrix form

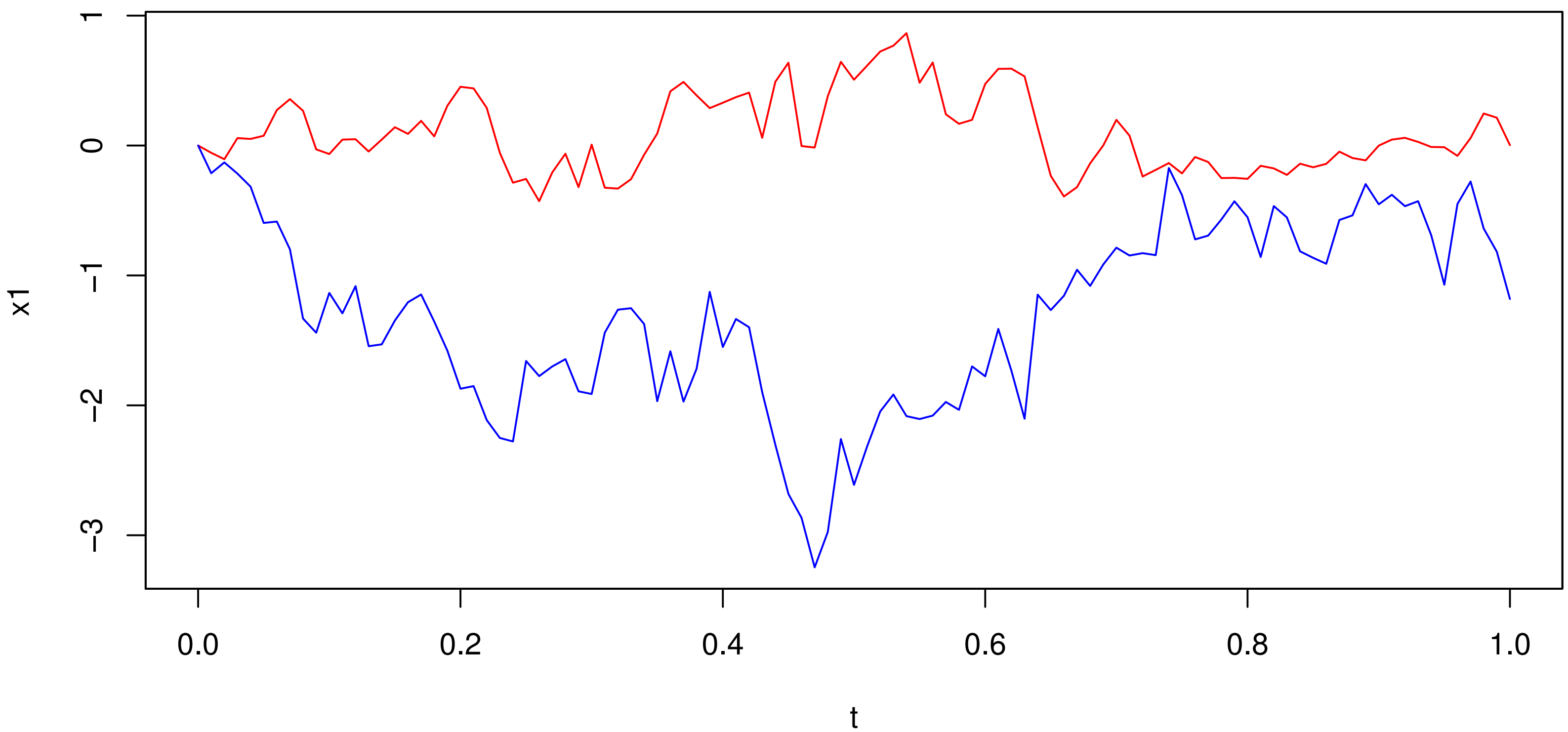
$$\begin{pmatrix} dX_t^1 \\ dX_t^2 \end{pmatrix} = \begin{pmatrix} -3X_t^1 \\ -X_t^1 - 2X_t^2 \end{pmatrix} dt + \begin{pmatrix} 1 & 0 & X_t^2 \\ X_t^1 & 3 & 0 \end{pmatrix} \begin{pmatrix} dW_t^1 \\ dW_t^2 \\ dW_t^3 \end{pmatrix}$$

```
> 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)
> mod3

Diffusion process
Number of equations: 2
Number of Wiener noises: 3
```

$$dX_t^1 = -3X_t^1 dt + dW_t^1 + X_t^2 dW_t^3$$
$$dX_t^2 = -(X_t^1 + 2X_t^2)dt + X_t^1 dW_t^1 + 3dW_t^2$$

```
> 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

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

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

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

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

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

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

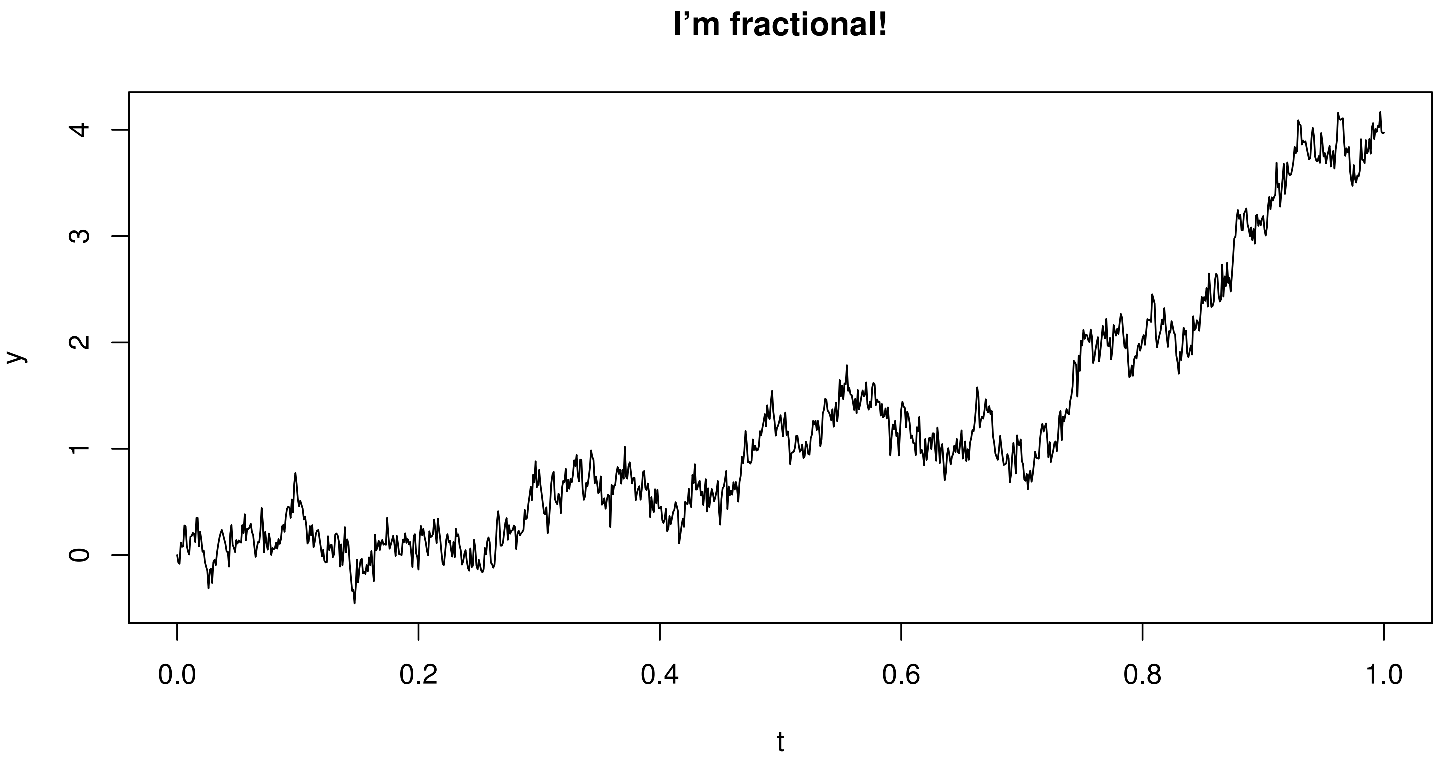
```
> mod4 <- setModel(drift="3*y", diffusion=1, hurst=0.3, solve.var="y")
```

The `hurst` slot is filled

```
> mod4

Diffusion process with Hurst index:0.30
Number of equations: 1
Number of Wiener noises: 1

> 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
  .. .. ..
  .. .. ..
  .. .. ..
  .. .. ..
  ..@ 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 $\mathrm{d}Y_t = 3Y_t\mathrm{d}t + \mathrm{d}W_t^H$


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

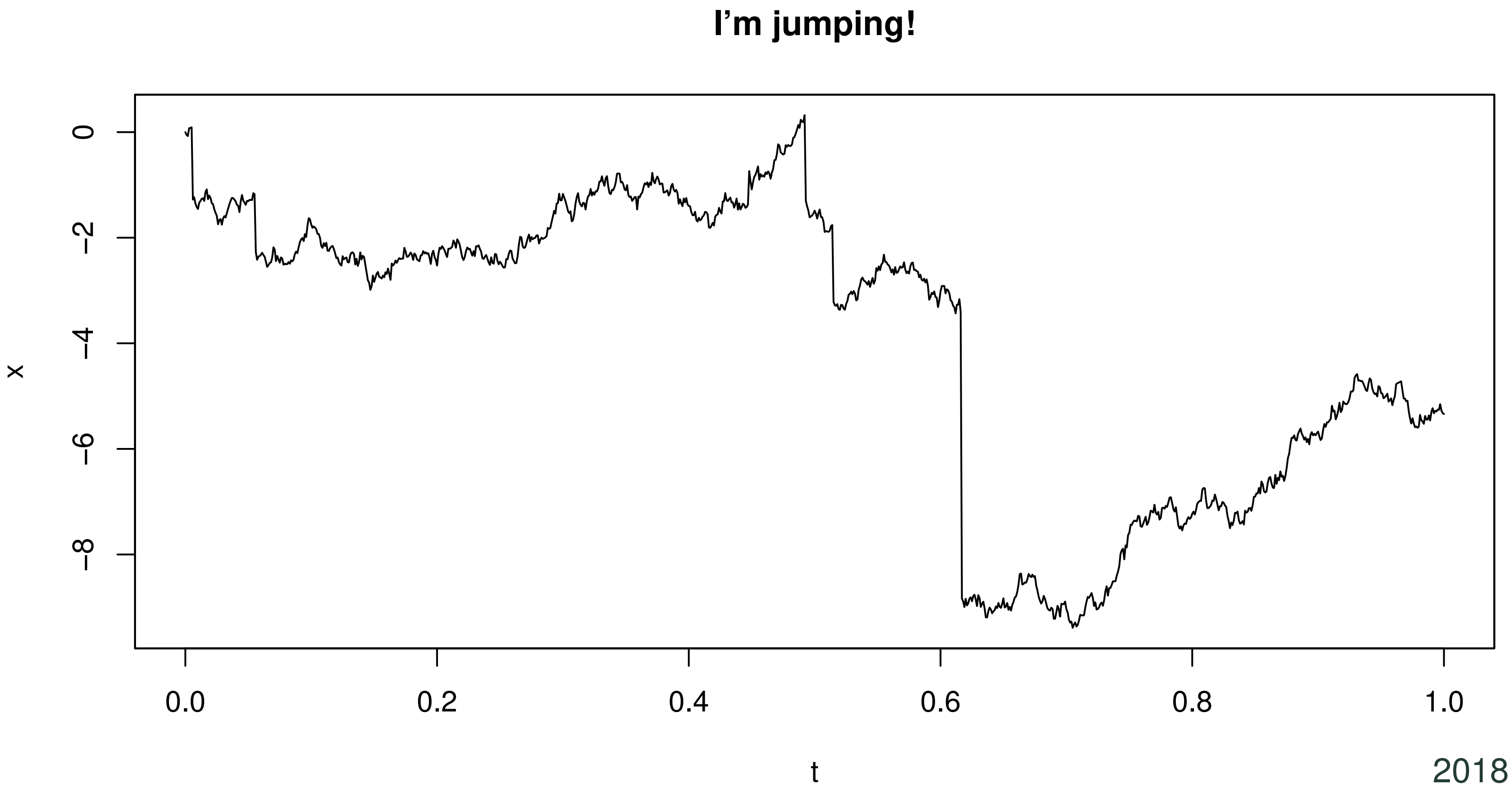
$$\mathrm{d}X_t = a(t, X_t, \theta)dt + b(t, X_t, \theta)\mathrm{d}W_t + c(t, X_t, \theta)\mathrm{d}Z_t$$

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)

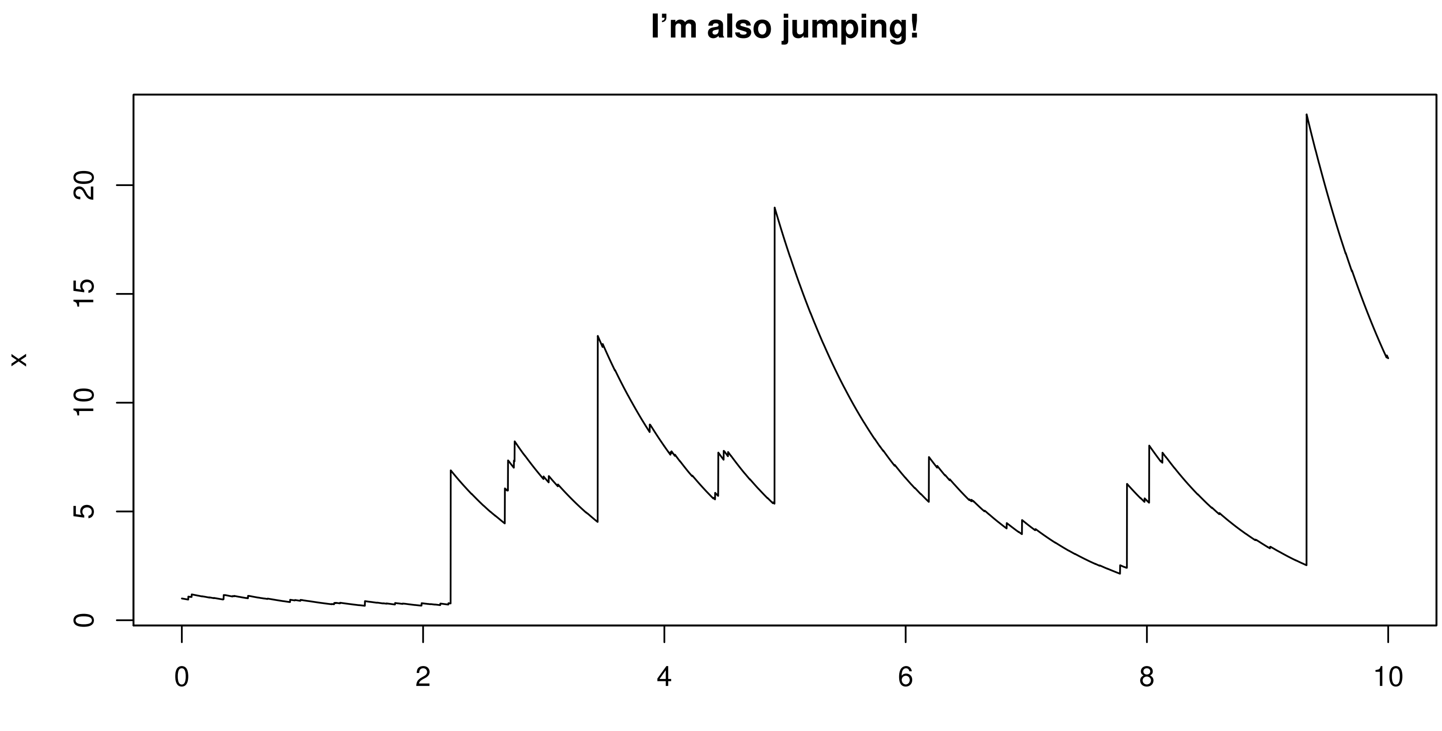
Jump process: $\mathrm{d}X_t = -\theta X_t \mathrm{d}t + \sigma \mathrm{d}W_t + Z_t$

```
> mod5 <- setModel(drift="-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!")
```



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

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

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

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

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

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

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

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

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.

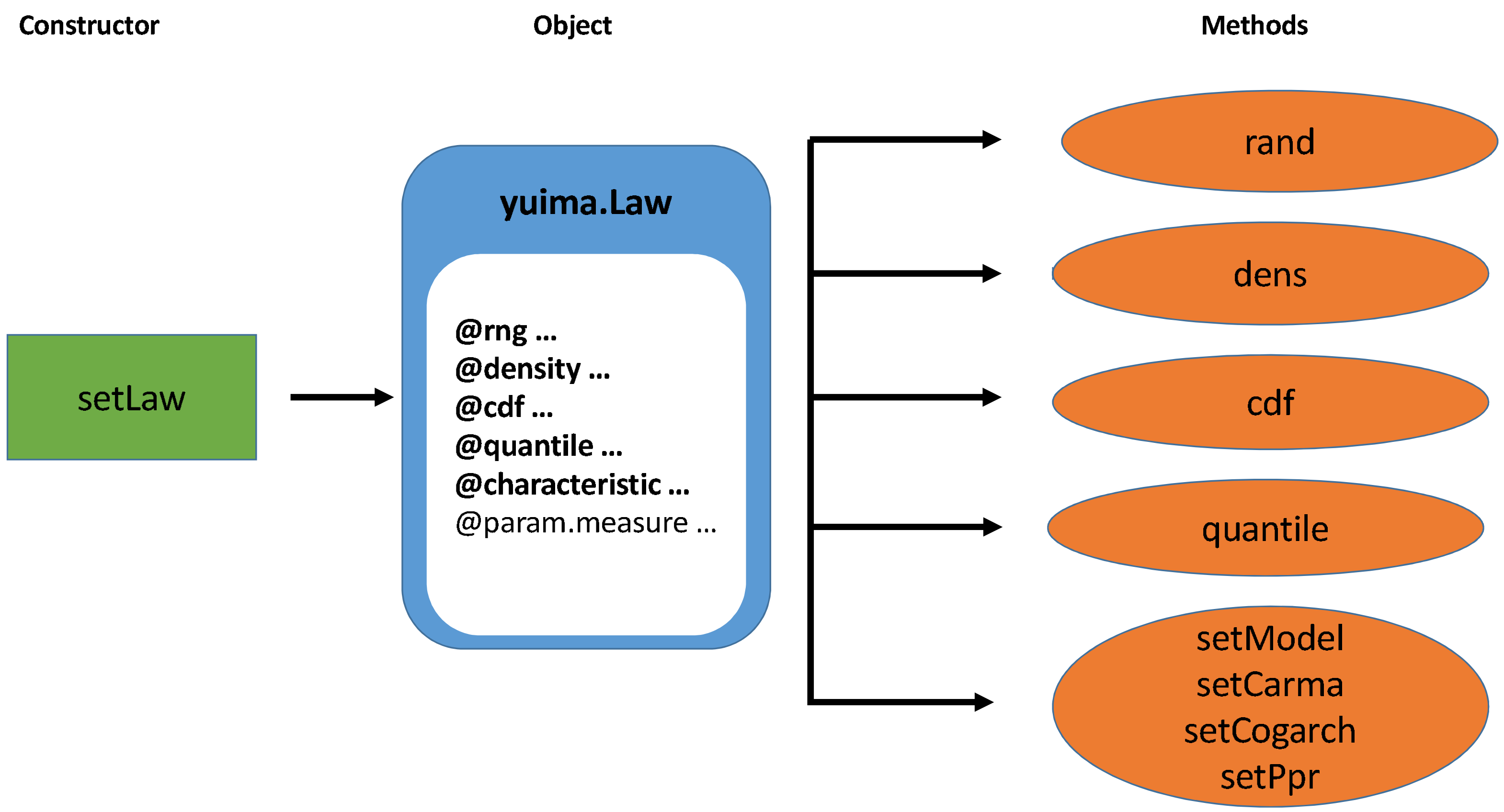
Models are specified via

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

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

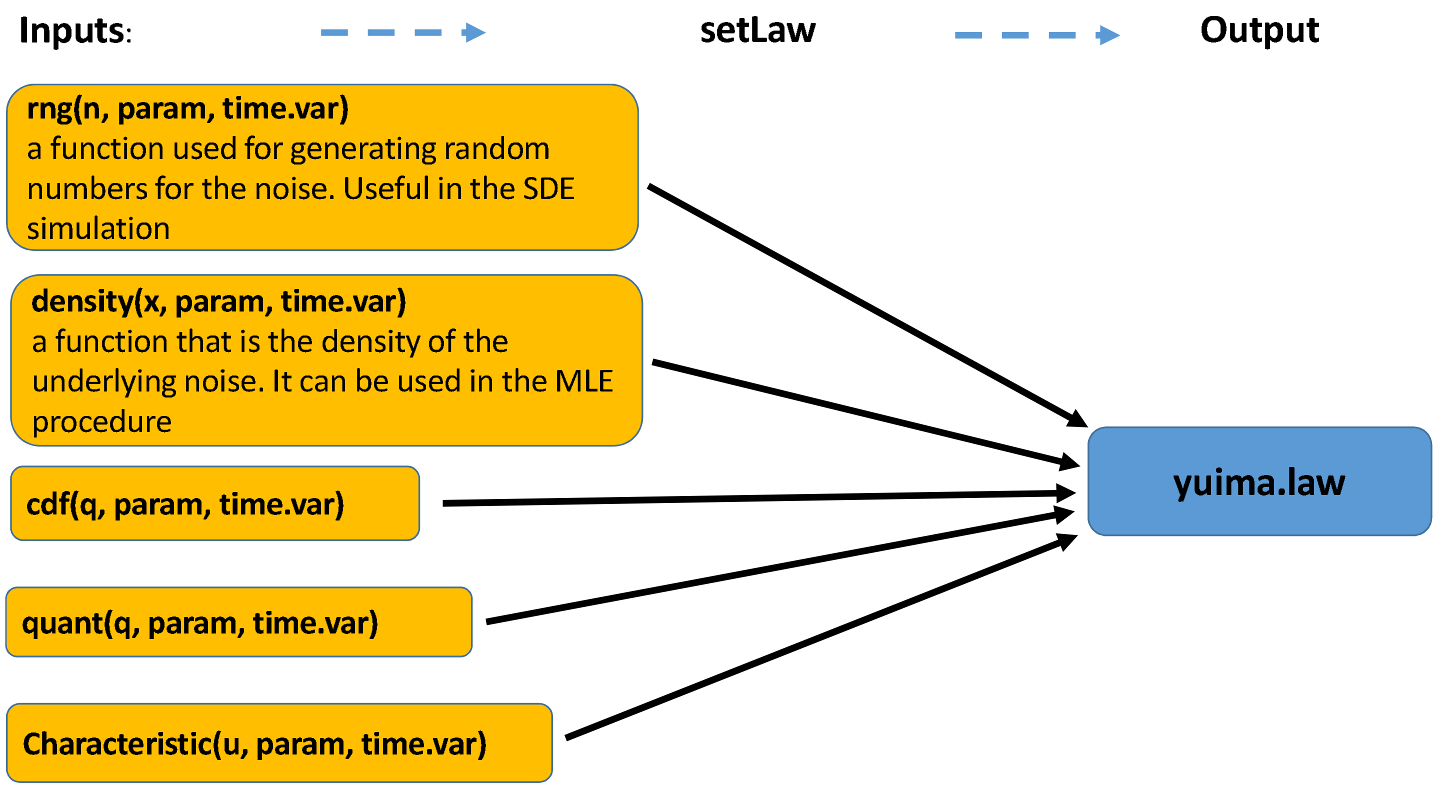
The package implements many multivariate RNG to simulate Lévy paths including `rIG`, `rNIG`, `rbgamma`, `rngamma`, `rstable`.
Other user-defined or packages-defined RNG can be used freely.

The `yuimaLaw` object is a mathematical description of probability models which include also the RNG, quantiles, characteristic function, etc.



YUIMA Law constructor

The `yuimaLaw` object is prepared with some constructor function `setLaw` where the user can specify all or part of its components.



YUIMA Law constructor

The `yuimaLaw` object is prepared with some constructor function `setLaw` where the user can specify all or part of its components; in the example, only the RNG has been specified:

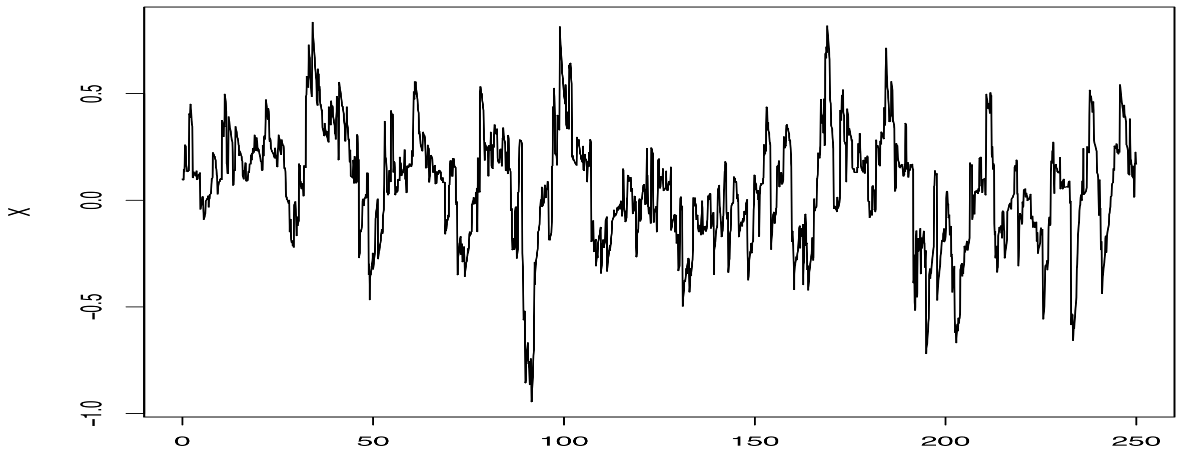
```
 > my.randMixedTS <- function(n, a,alpha, lambda_p, lambda_m, t {
  par <- setMixedTS.param(mu0 = 0, mu = 0, sigma = 1, a = a * t,
          alpha = alpha, lambda_p = lambda_p, lambda_m = lambda_m)
  rand <- rMixedTS(object = par, x=n)@Data
  return(as.matrix(rand))
}
> my.L <- setLaw(rng = my.randMixedTS)
```

Suppose we want to specify and simulated this model:

$$\mathrm{d}X_t = 0.4(0.1 - X_t)\mathrm{d}t + 0.2\mathrm{d}L_t$$

where L_t has a mixed tempered stable distribution.

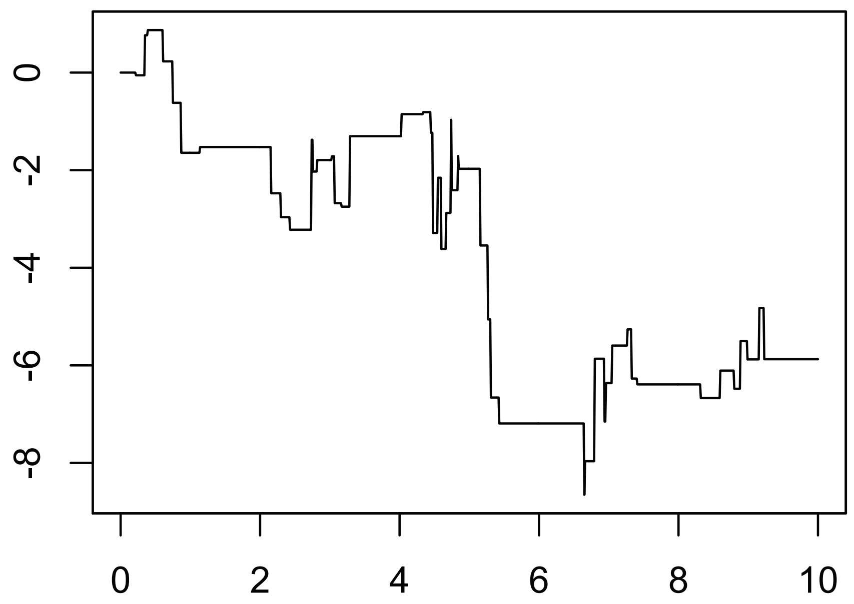
```
> mod1 <- setModel(drift = c("0.4*(0.1-X)"), diffusion = c("0"), jump.coeff = c("0.2"),
  measure = list(df = my.L), measure.type = c("code"), solve.variable = c("X"), xinit=c("0.1"))
> par <- par <- list(a = 1.5, alpha = 1.5, lambda_p = 1, lambda_m = 1)
> sim1 <- simulate(object = mod1, true.parameter = par,
  sampling = setSampling(0,250, n = 2500))
> plot(sim1)
```

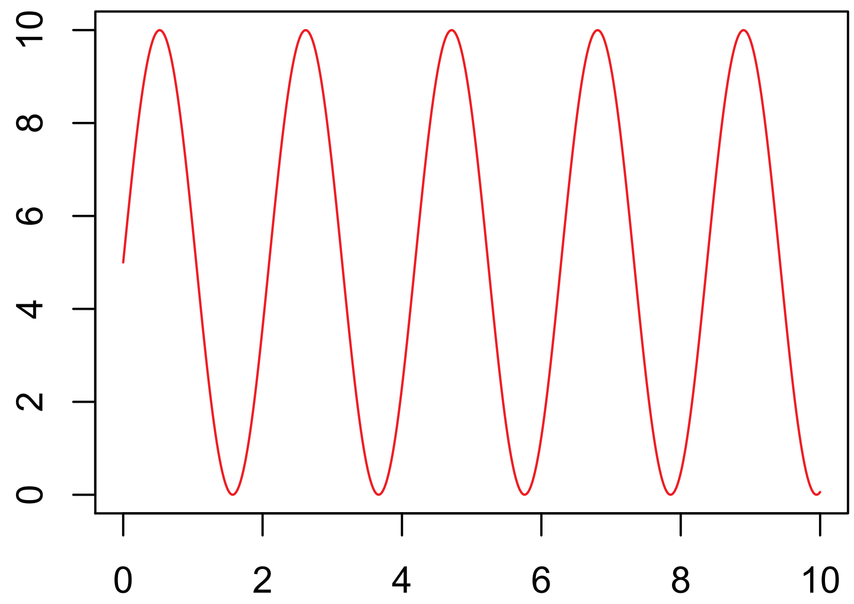


Compound Poisson Processes

There is a simplified way to specify directly Compound Poisson Processes using `setPoisson`. The next code defines and simulates an inhomogeneous Compound Poisson Process with Gaussian jumps:

```
> samp <- setSampling(Terminal=10, n=1000)
> mod7 <- setPoisson(intensity="beta*(1+sin(lambda*t))", df=list("dnorm(z,0,1)"))
> f <- function(t) beta*(1+sin(lambda*t))
> set.seed(123); lambda <- 3;  beta <- 5
> y7 <- simulate(mod7, true.par=list(lambda=lambda, beta=beta),sampling=samp)
> plot(y7);  curve(f, 0, 10, col="red",n=500)
```





Let N_t is a Poisson process, with $\Lambda_t = \int_0^t \lambda(s)\mathrm{d}s$. Assume further that $\lambda(t, N_t, X_t)$ is stochastic and dependent also on the process itself (feedback effect) and some other covariate process X_t (regression model).
An Hawkes process Y_t is a Point Process Regression (PPR) Model consisting of

$$Y_t = [X_t, N_t]^T$$

More precisely, the intensity function of N_t can be described as

$$\begin{aligned}\lambda_t &= g(t, Y_t, \theta) + \int_{t_0}^{t^-} K(t-s, Y_t, \theta)\ \mathrm{d}Y_s \\ &= g(t, [X_t, N_t]^T, \theta) + \int_{t_0}^{t^-} K(t-s, [X_t, N_t]^T, \theta)\ \mathrm{d}\begin{pmatrix} X_s \\ N_s \end{pmatrix}\end{aligned}$$

The function $g(t, Y_t, \theta)$ is a non negative predictable process $K(t-s, Y_s, \theta)$ is a non-negative

predictable matrix process.

$$Y_t = [r_t, Q_t, N_t]^T$$

$$\lambda_t = \exp\left(\mu_0 + \mu_1 \cdot \ln(1 + r_t) + \mu_2 \cdot \ln(1 + Q_t)\right) + \int_0^t c \cdot e^{-a\cdot(t-s)} \mathrm{d}N_s$$

```
> my.rMTY <- function(n,t){
 res0 <- t(t(rgamma(n, 0.1*t)))
 res1 <- t(t(rgamma(n, 0.1*t)))
 res2 <- t(t(rep(1,n)))
 res <- cbind(res0,res1,res2)
 return(res)
}
> Law.MTY <- setLaw(rng = my.rMTY) # we prepare the law
# we prepare the covariate process
> modMTY <- setModel(drift = c("0.4*(0.1-Q)",".4*(0.1-R)","0"), diffusion = c("0","0","0"),
  jump.coeff = matrix(c("1","0","0","0","1","0","0","0","1"),3,3),
  measure = list(df = Law.MTY), measure.type = c("code","code","code"),
  solve.variable = c("Q","R","N"), xinit=c("0.25","0.25","0"))
```

$$Y_t = [r_t, Q_t, N_t]^T$$

$$\lambda_t = \exp\left(\mu_0 + \mu_1 \cdot \ln(1 + r_t) + \mu_2 \cdot \ln(1 + Q_t)\right) + \int_0^t c \cdot e^{-a\cdot(t-s)} \mathrm{d}N_s$$

```
> gFun <- "exp(mu0 + mu1*log(1+R)+mu2*log(1+Q))"
> Kernel <- "c*exp(-a*(t-s))"

# definition of Hawkes process
> prvMTY <- setPpr(yuima = modMTY, counting.var="N", gFun=gFun, Kernel = as.matrix(Kernel),
  lambda.var = "lambda", var.dx = "N", lower.var="0", upper.var = "t")
```

Let Z_t is a Lévy process and $p,\ q$ non-negative integers such that $p > q \geq 0$.

The CARMA(p,q) process (see Brockwell, 2001) is defined as:

$$a(D)Y_t = b(D)DZ_t \tag{1}$$

D is the differentiation operator with respect to t while $a(\cdot)$ and $b(\cdot)$ are two polynomials:

$$a(u) = u^p + a_1 u^{p-1} + \cdots + a_p$$

$$b(u) = b_0 + b_1 u^1 + \cdots + b_{p-1} u^{p-1}$$

where $a_1; \cdots ; a_p$ and $b_0, \cdots , b_q$ are coefficients such that $b_q \neq 0$ and $b_j = 0\ \forall j > q$.

CARMA(p,q): state-space representation

It is more convenient to use the following state space representation of a CARMA(p,q) model

$$Y_t = \mathbf{b}^\intercal X_t$$

where X_t is a vector process of dimension p satisfying the following system of stochastic differential equations:

$$\mathrm{d}X_t = AX_t\mathrm{d}t + \mathbf{e}\mathrm{d}Z_t$$

with A is the $p \times p$ matrix defined as:

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_p & -a_{p-1} & -a_{p-2} & \dots & -a_1 \end{bmatrix}$$

The $p \times 1$ vectors $\mathbf{e}$ and $\mathbf{b}$ are respectively: $\mathbf{e} = [0, \dots, 0, 1]^\intercal$ and $\mathbf{b} = [b_0, \dots, b_{p-1}]^\intercal$.

Back to CARMA: the `yuima.carma` class

An object of the class `yuima.carma` contains all informations related to a general linear state space model that encompasses the CARMA model described in the previous slides. The mathematical description of this general model is given by the following system of equations:

$$\begin{aligned} Y_t &= \mu + \sigma \cdot \mathbf{b}^\intercal X_t \\ \mathrm{d}X_t &= AX_t \mathrm{d}t + \mathbf{e}\left(\gamma_0 + \gamma^\intercal X_t\right) \mathrm{d}Z_t \end{aligned} \tag{2}$$

where $\mu \in \mathbb{R}$ and $\sigma \in (0, +\infty)$ are location and scale parameters respectively. The vector $\mathbf{b} \in \mathbb{R}^p$ contains the moving average parameters $b_0, b_1, \ldots, b_q \neq 0, b_{q+1} = \cdots b_{p-1} = 0$ while the A is a $p \times p$ matrix whose last row contains the autoregressive parameters $a_1, \ldots, a_p$; A and $\mathbf{e}$ are as before.

The `setCARMA` function

We use the constructor `setCARMA` for building an object of class `yuima.carma`.

The arguments used in a call to the constructor `setCARMA()` are:

```
setCARMA(p,q,loc.par=NULL,scale.par=NULL,ar.par="a",ma.par="b",
lin.par=NULL,CARMA.var="v",Latent.var="x",XinExpr=FALSE, ...)
```

- `p` is a integer number the indicates the dimension of autoregressive coefficients.
- `q` is the dimension of moving average parameters.
- `XinExpr` is a logical variable. If `XinExpr=FALSE`, the starting condition of X_t is zero otherwise each component of X_t has a parameter as a starting point.

By default `setCARMA` build a CARMA model driven by a standard Brownian motion.
The `dots` arguments are used to pass information when the underlying noise is a (jump) Lévy process. In particular the following two arguments are necessary

- `measure` Lévy measure of jump variables.
- `measure.type` type specification for Lévy measure. `"CP"` for compound poisson, `"code"` for other Lévy processes such as Inverse Gaussian, Normal Inverse Gaussian, Gamma, Variance Gamma, Bilateral Gamma and etc.

For an object of `yuima.carma` methods for simulation (`simulate`) and for estimation (`qmle`) are available and they are based on the state space representation.

The setCARMA function

Assume that we want to build a CARMA(p=3,q=1) model driven by a standard Brownian Motion with location parameter. In this case, the state space model in (2) can be written in a explicit way as follows:

$$\begin{aligned} Y_t &= b_0 X_{0,t} + b_1 X_{1,t} \\ \mathrm{d}X_{0,t} &= X_{1,t}\mathrm{d}t \\ \mathrm{d}X_{1,t} &= X_{2,t}\mathrm{d}t \\ \mathrm{d}X_{2,t} &= \left[-a_3 X_{0,t} - a_2 X_{1,t} - a_1 X_{2,t}\right]\mathrm{d}t + dZ_t \end{aligned}$$

where $Z_t = W_t$ is a Wiener process. For this reason, we instruct `yuima` to create an object of class `yuima.carma` using the code listed below.

```
> 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 8 parameters
```

The `setCARMA` function

The CARMA(3,1) model is represented internally in `yuima` as:

$$\mathrm{d}\begin{bmatrix} Y_t \\ X_{0,t} \\ X_{1,t} \\ X_{2,t} \end{bmatrix} = \begin{bmatrix} b_0 X_{0,t} + b_1 & X_{1,t} & \\ & X_{1,t} & \\ & & X_{2,t} \\ -a_3 X_{0,t} - a_2 & X_{1,t} - a_1 & X_{2,t} \end{bmatrix} \mathrm{d}t + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \mathrm{d}Z_t \qquad (3)$$

Notice that, since we define the CARMA(p,q) model using the standard `yuima` mathematical description, we need to rewrite the observable process Y_t as a stochastic differential equation. The location parameter c_0 is contained in the slot `xinit` where the starting condition of the variable Y_t is:

$$Y_0 = c_0 + b_0 X_{0,0} + b_1 X_{1,0}$$

To ensure the existence of a second order solution, we choose the autoregressive coefficients $\mathbf{a} := [a_1, a_2, a_3]$ such that the eigenvalues of the matrix A are real and negative. Indeed, $a_1 = 4$, $a_2 = 4.75$ and $a_3 = 1.5$, it is easy to verify that the eigenvalues of matrix A are $\lambda_1 = -0.5$, $\lambda_2 = -1.5$ and $\lambda_3 = -2$.

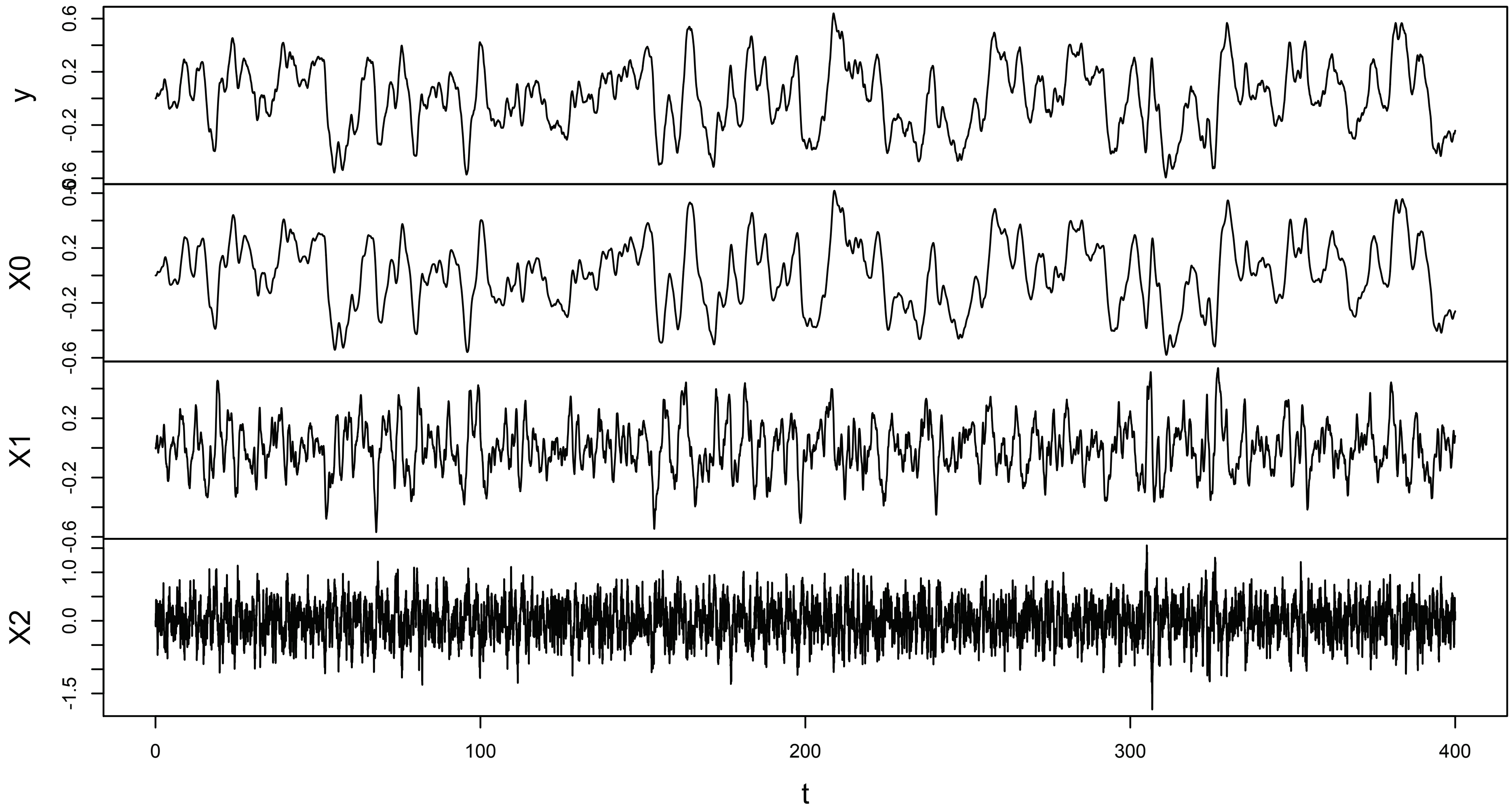
We now set the parameters, prepare the sampling scheme and simulate a trajectory of the CARMA process

```
> par.carma <- list(a1=4,a2=4.75,a3=1.5,b0=1,b1=0.23,c0=0)
> samp <- setSampling(Terminal=400, n=16000)
> set.seed(123)
> carma <-simulate(carma.mod, true.parameter=par.carma, sampling=samp)
```

we can now plot the simulated trajectory

```
> plot(carma)
```

An example of CARMA(3,1) trajectory



Example of CARMA(2,1) and VG jumps

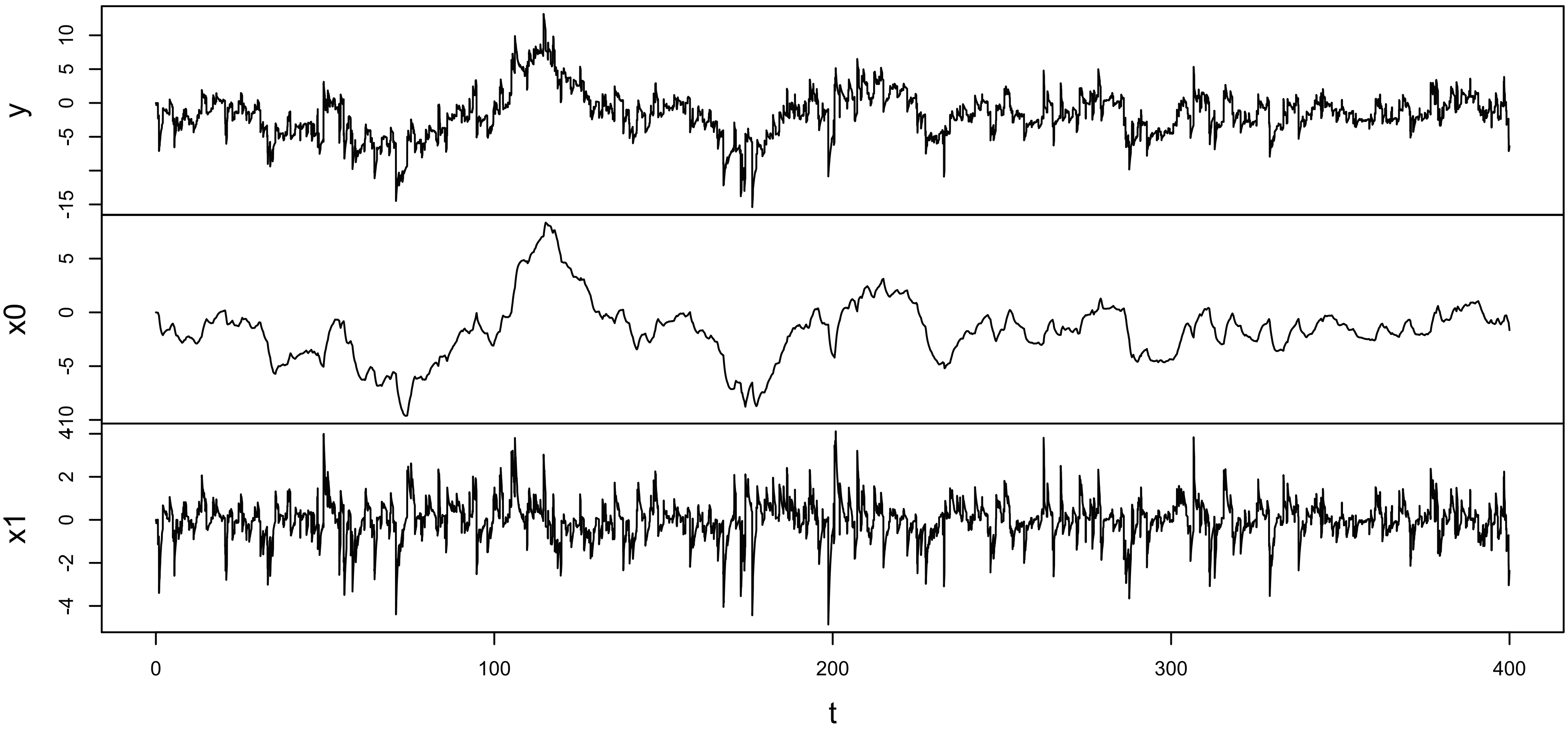
In this case, the underlying Lévy is a Variance Gamma model (Madan, 1990). We setup the model as follows

```
> modVG <- setCARMA(p=2,q=1,CARMA.var="y",
       measure=list("rngamma(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)
> set.seed(100)
> simVG <- simulate(modVG, true.parameter=true.parmVG,  sampling=samp)
> plot(simVG)
```

we can now plot the simulated trajectory

```
> plot(simVG)
```

An example of CARMA(2,1) & VG trajectory



The COGARCH(p,q) process, introduced in Brockwell *et al* (2006) is defined as:

$$\begin{cases} \mathrm{d}G_t = \sqrt{V_t}\mathrm{d}Z_t \\ V_t = a_0 + \mathbf{a}^\top Y_{t-} \\ \mathrm{d}Y_t = AY_{t-}\mathrm{d}t + \mathbf{e}\left(a_0 + \mathbf{a}^\top Y_{t-}\right)\mathrm{d}\left[Z,Z\right]_t^d \end{cases} \tag{4}$$

where $q \geq p \geq$, $Y_t = \left[Y_{1,t},\ldots,Y_{q,t}\right]^\top$, $\mathbf{a} = \left[a_1,\ldots,a_p,a_{p+1},\ldots,a_q\right]^\top$ with $a_{p+1} = \cdots = a_q = 0$,

$$A = \begin{bmatrix} 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \\ -b_q & -b_{q-1} & \ldots & -b_1 \end{bmatrix}.$$

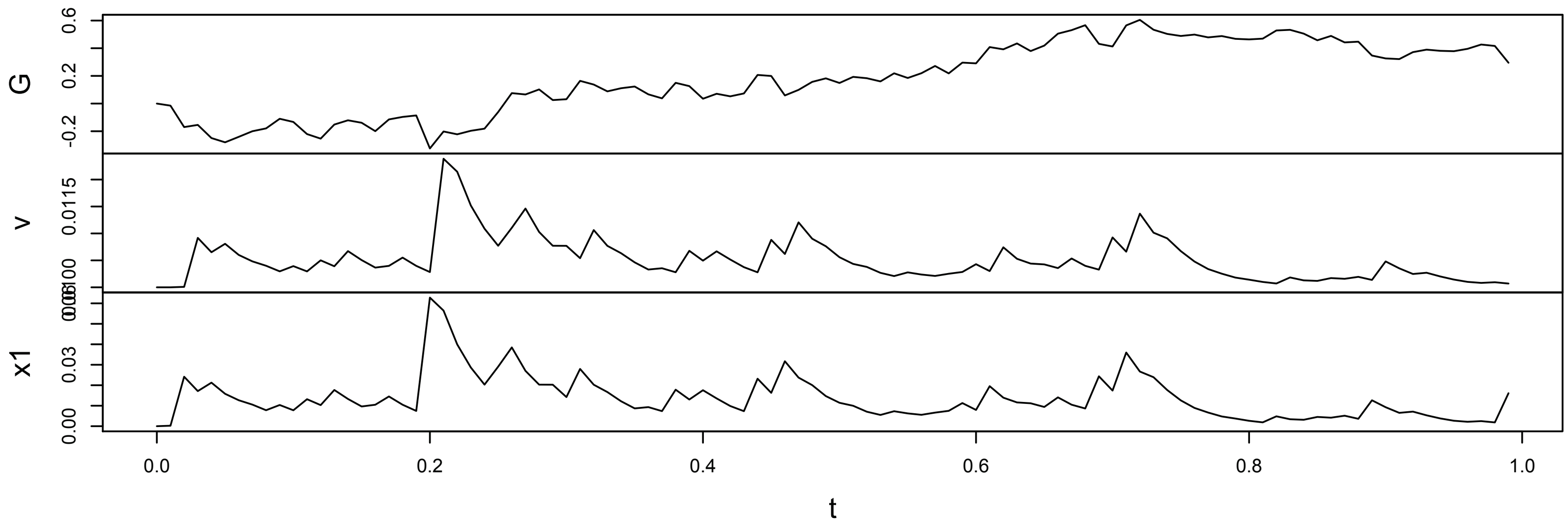
$\mathbf{e} \in R^q$ contains zero entries except for the last component that is equal to one and

$$\left[Z,Z\right]_t^d := \sum_{0\leq s\leq t}\left(\Delta Z_s\right)^2. \tag{5}$$

is the discrete part of the quadratic variation of the underlying Lévy process.

The constructor function is called `setCogarch` and it is quite similar to `setCogarch`. Suppose se want to define a COGARCH(1,1) model with a Variance Gamma Lévy noise. We proceed as follows:

```
> model1 <- setCogarch(p = 1, q = 1,
+   measure = list(df="rvgamma(z, 100, 2, .1, .1)"),
+    measure.type = "code", Cogarch.var = "G", V.var = "v",
+    Latent.var = "x", XinExpr = TRUE)
> param1 <- list(a1 = 0.038, b1 = 30, a0 = 0.01, x01 = 0)
>  set.seed(123);  plot( simulate(model1, true.parameter = param1) )
```



Inference for Stochastic Processes and much more

For all the models presented so far, there exists the `qmle` for Quasi-;aximum Likelihood estimation which has an interface similar to the standard `mle` with the only difference that instead of a likelihood function, the input is one of the above `yuima` models.

The only exception is the fractional Gaussian case which makes use of `gmm`-type approach.

Hypotheses testing, AIC, Adaptive Bayes Estimation, Change Point analysis and much more have been developed for general SDEs.

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

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

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.

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
```

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. $W_t = (W_t^1, \dots, W_t^r)$.

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

$$dX_t^\varepsilon = \mu X_t^\varepsilon dt + \varepsilon X_t^\varepsilon dW_t$$

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

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

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

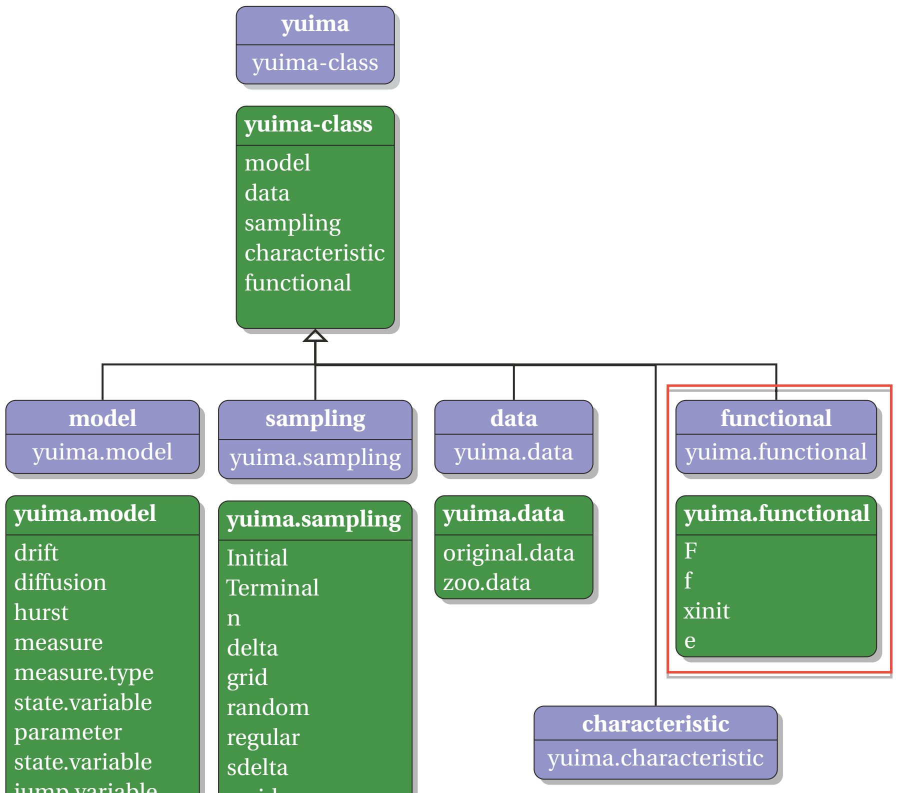
$$\max(x - K, 0)$$

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)
```

```
> 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
```

Then, it is as easy as

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

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
```

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 $F(X^{1,\varepsilon}, X^{2,\varepsilon})$.

LASSO estimation & model selection

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.

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)	$\mathrm{d}X_t = \alpha \mathrm{d}t + \sigma \mathrm{d}W_t$		0	0
Vasicek (1977)	$\mathrm{d}X_t = (\alpha + \beta X_t)\mathrm{d}t + \sigma \mathrm{d}W_t$			0
Cox, Ingersoll and Ross (1985)	$\mathrm{d}X_t = (\alpha + \beta X_t)\mathrm{d}t + \sigma\sqrt{X_t}\mathrm{d}W_t$			$1/2$
Dothan (1978)	$\mathrm{d}X_t = \sigma X_t \mathrm{d}W_t$	0	0	1
Geometric Brownian Motion	$\mathrm{d}X_t = \beta X_t \mathrm{d}t + \sigma X_t \mathrm{d}W_t$	0		1
Brennan and Schwartz (1980)	$\mathrm{d}X_t = (\alpha + \beta X_t)\mathrm{d}t + \sigma X_t \mathrm{d}W_t$			1
Cox, Ingersoll and Ross (1980)	$\mathrm{d}X_t = \sigma X_t^{3/2} \mathrm{d}W_t$	0	0	$3/2$
Constant Elasticity Variance	$\mathrm{d}X_t = \beta X_t \mathrm{d}t + \sigma X_t^{\gamma} \mathrm{d}W_t$	0		
CKLS (1992)	$\mathrm{d}X_t = (\alpha + \beta X_t)\mathrm{d}t + \sigma X_t^{\gamma} \mathrm{d}W_t$			

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)

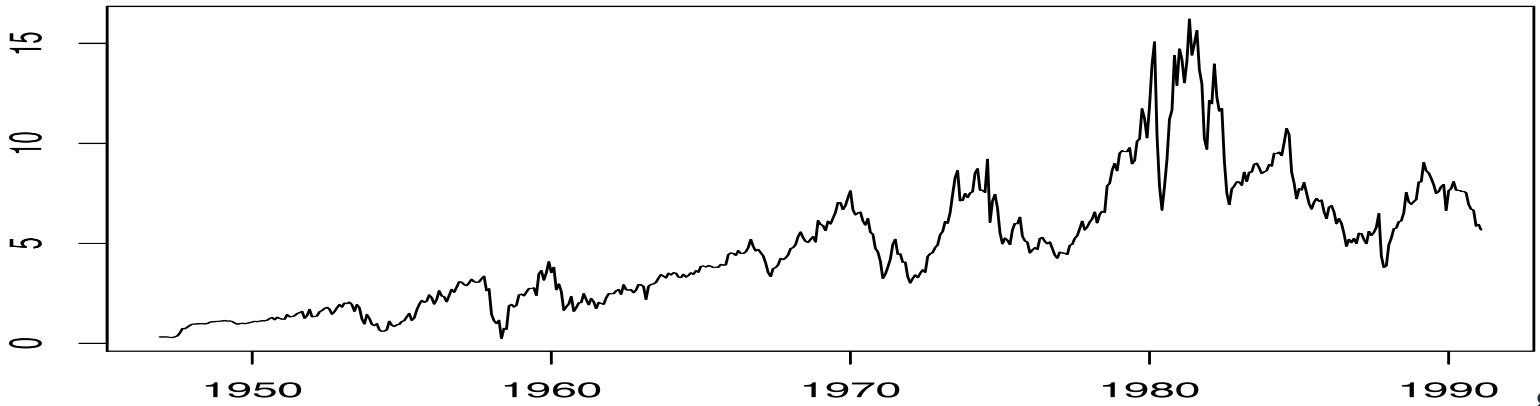
LASSO selected: Cox, Ingersoll and Ross (1980) model

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

An example of Lasso use on real data with CKLS model

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

```
> 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.

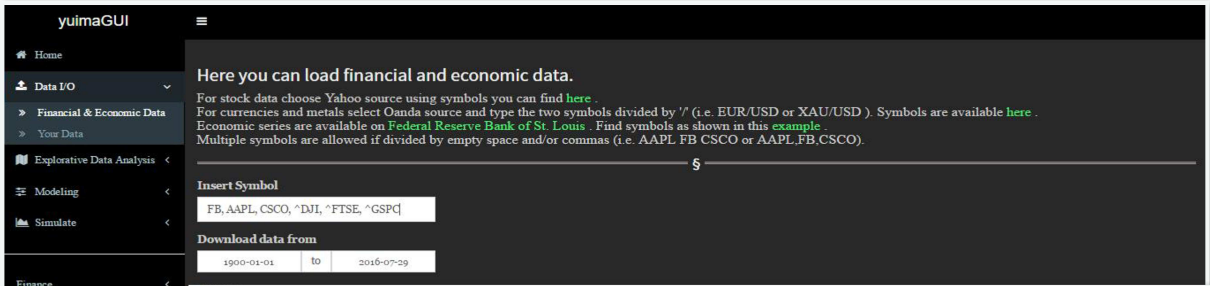

```
> lambda0 <- list(alpha=10, beta =10, sigma =10, gamma =10)
> start <- list(alpha=1, beta =-.1, sigma =.1, gamma =1)
> low <- list(alpha=-5, beta =-5, sigma =-5, gamma =-5)
> upp <- list(alpha=8, beta =8, sigma =8, gamma =8)
> lasso10 <- lasso(yuima, lambda0, start=start, lower=low, upper=upp,
   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
```

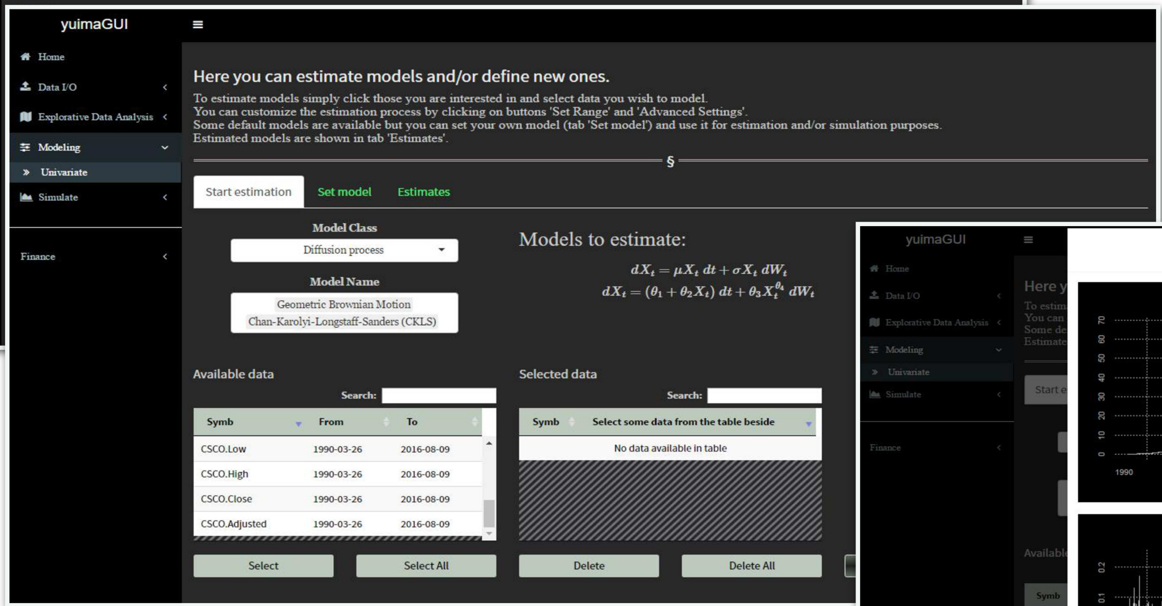
$$\mathrm{d}X_t = (\alpha + \beta X_t)\mathrm{d}t + \sigma X_t^{\gamma}\mathrm{d}W_t$$

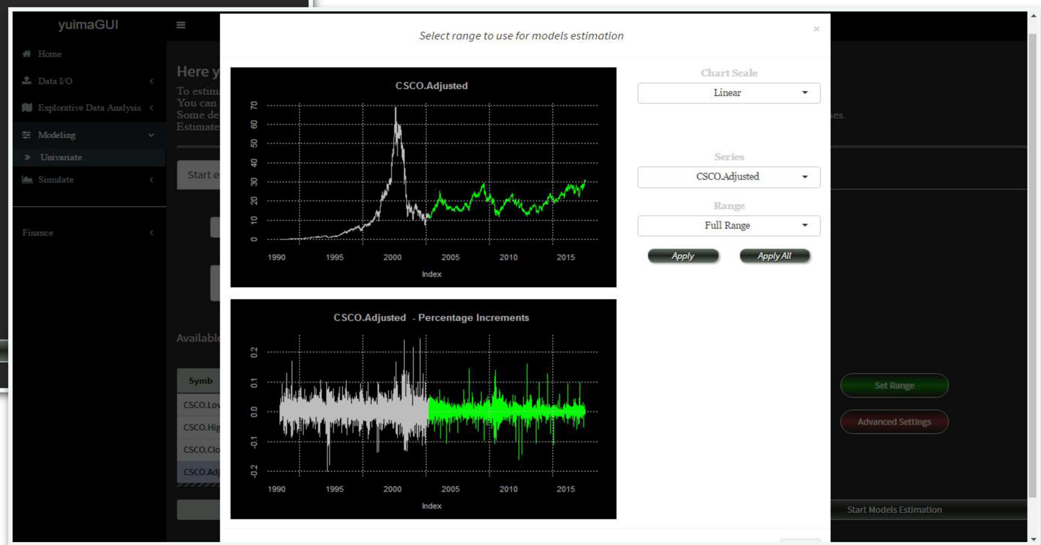
$$\mathrm{d}X_t = 0.6\mathrm{d}t + 0.12 X_t^{\frac{3}{2}}\mathrm{d}W_t$$

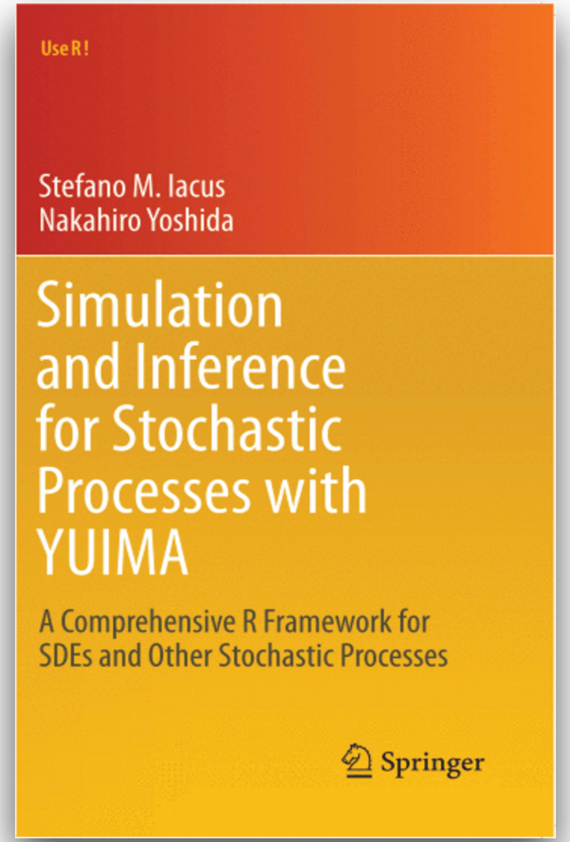
For further information

http://www.yuima-project.com









yuimaGUI is also available!

Emanuele Guidotti Stefano M. Iacus Lorenzo Mercuri

yuimaGUI

A graphical user interface for computational finance based on the `yuima` R package

Complexity Level

Deep scientific knowledge

↓

yuima R package
(and others)

↓

yuimaGUI

Research level

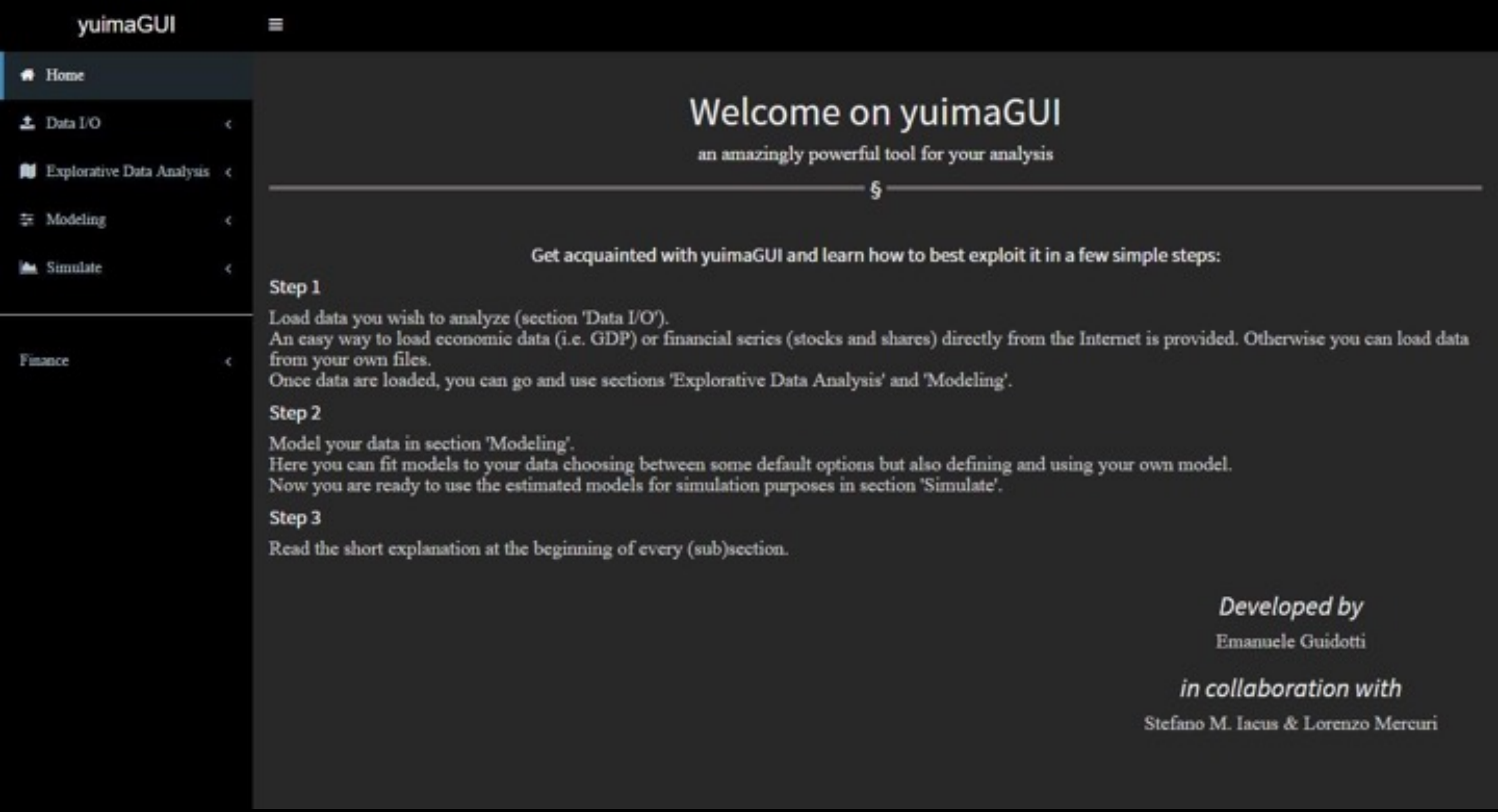
↓

Programming level

↓

User-friendly level

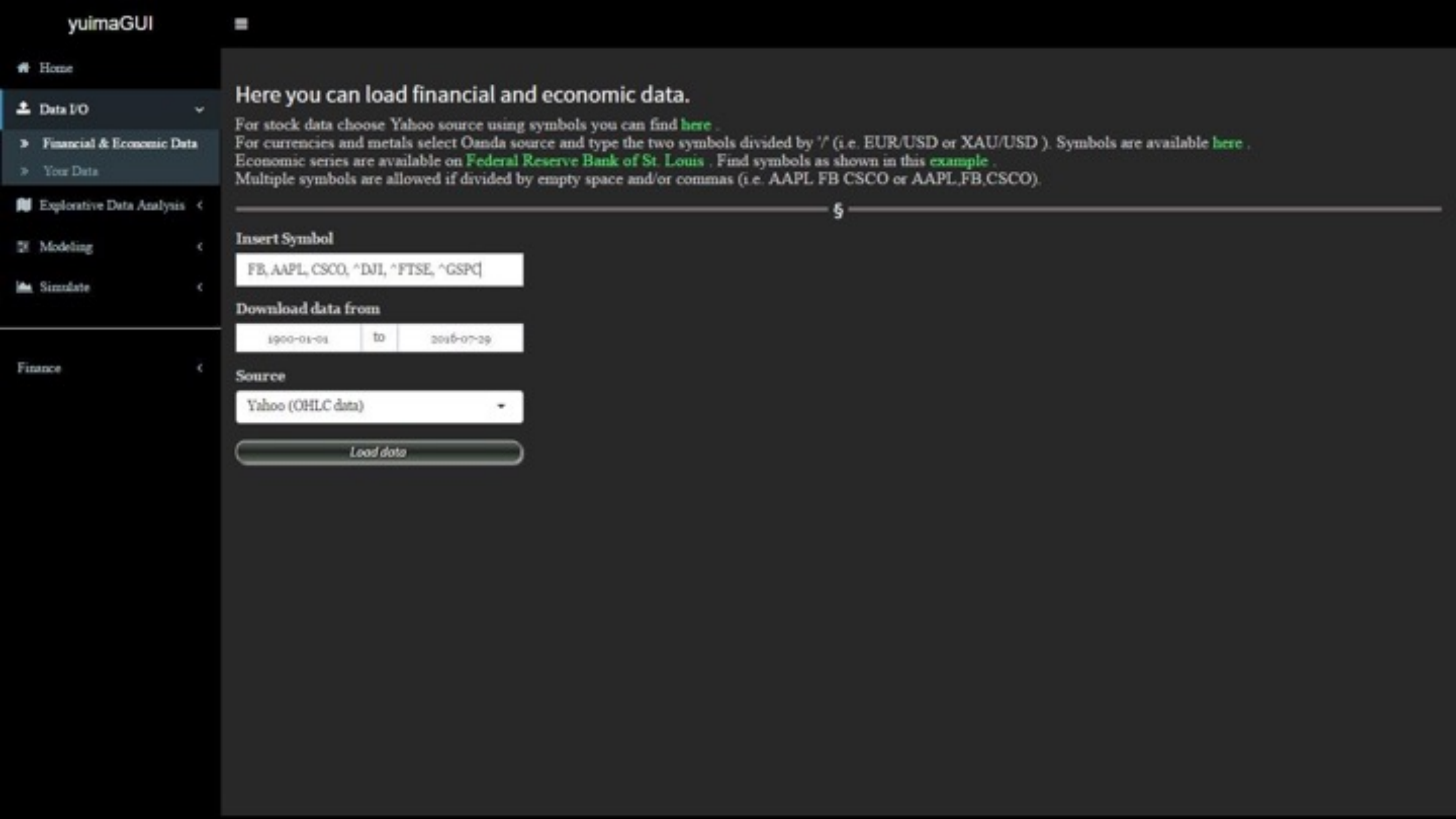
Typical Usage of yuimaGUI



Loading Data → Explorative Data Analysis → Modeling → Simulation

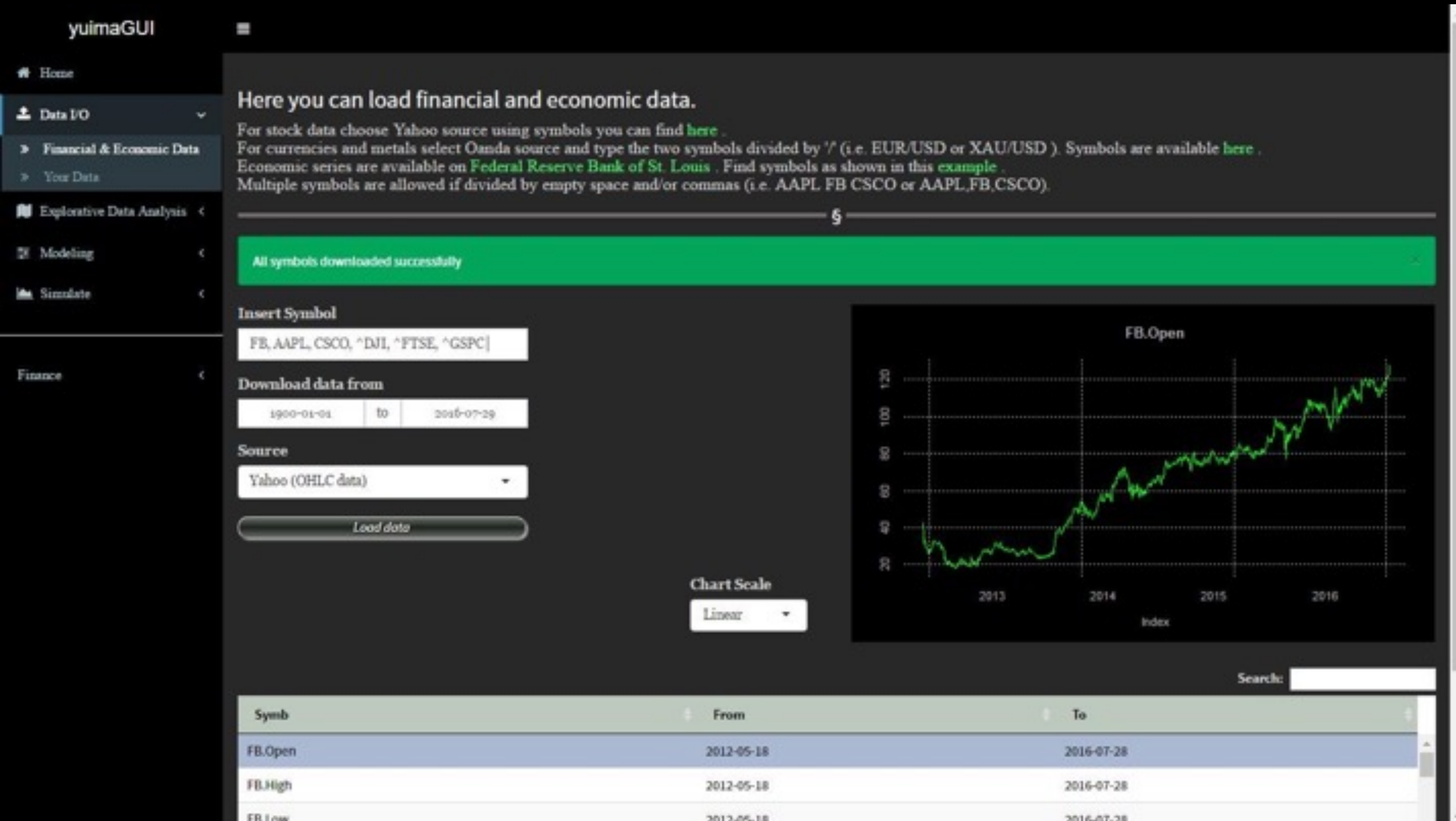
Loading Data → Explorative Data Analysis → Modeling → Simulation

Financial & Economic Data



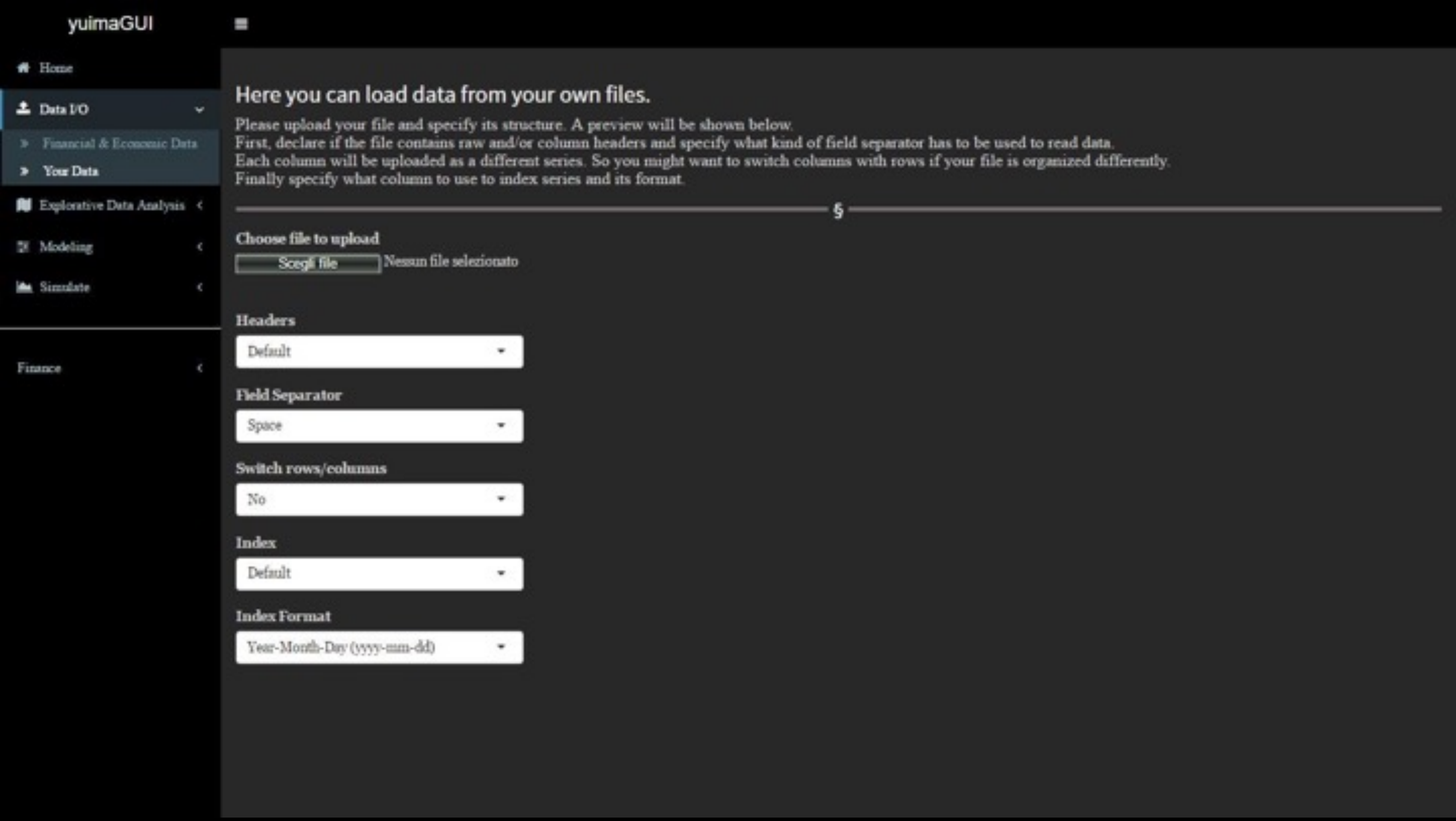
Loading Data → Explorative Data Analysis → Modeling → Simulation

Financial & Economic Data



Loading Data → Explorative Data Analysis → Modeling → Simulation

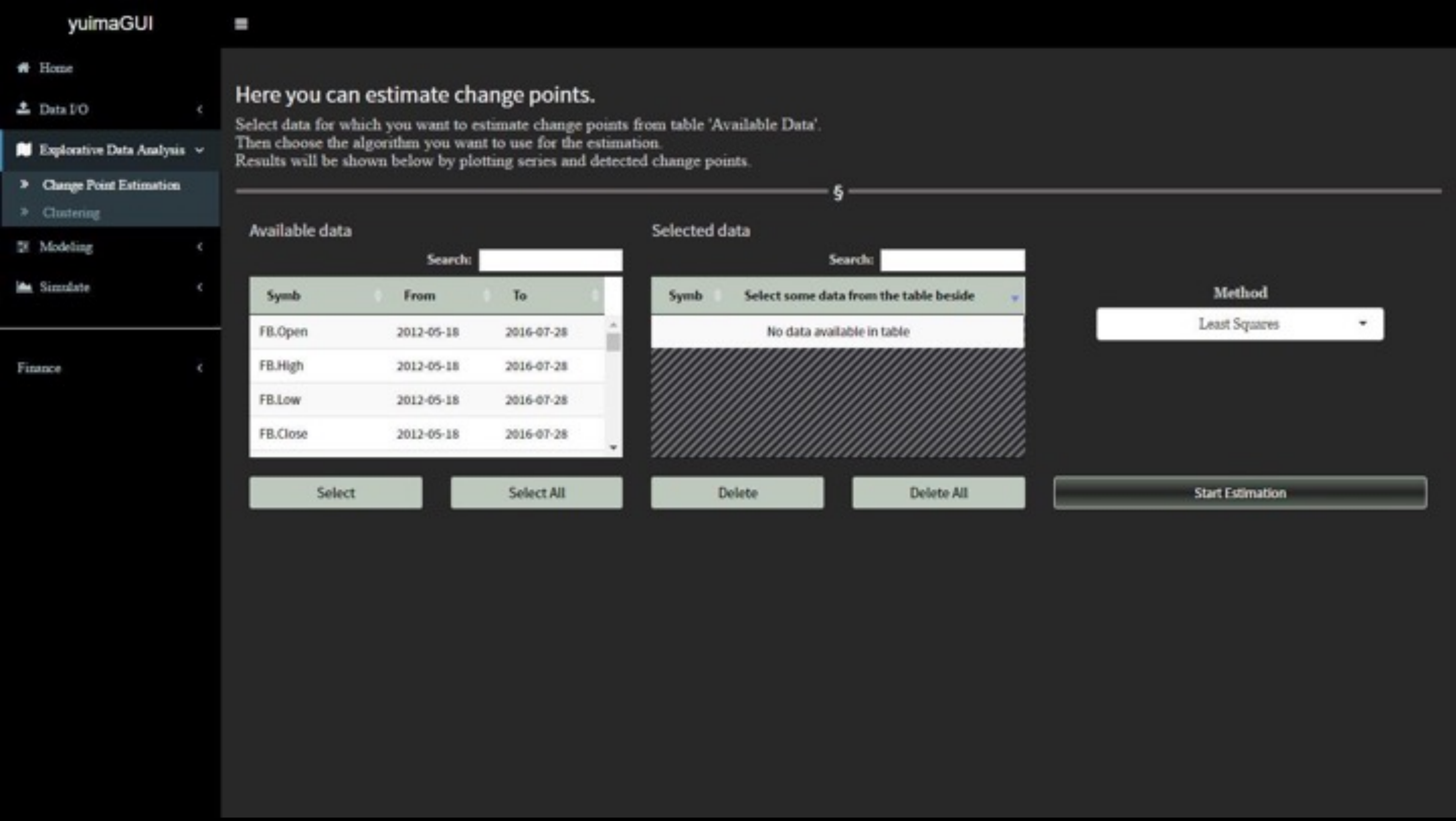
Your Data



Loading Data → Explorative Data Analysis → Modeling → Simulation

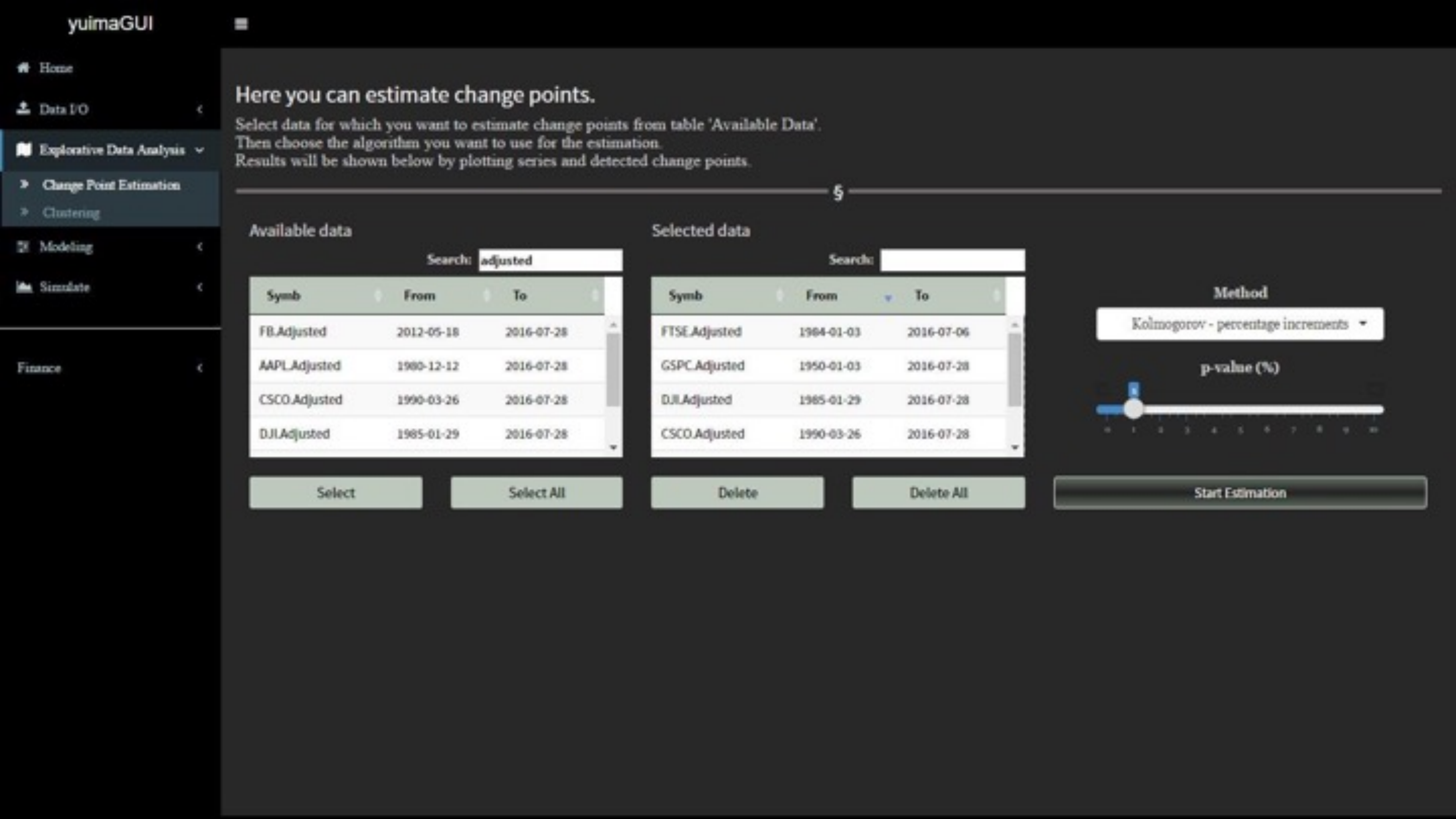
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Change Point Estimation



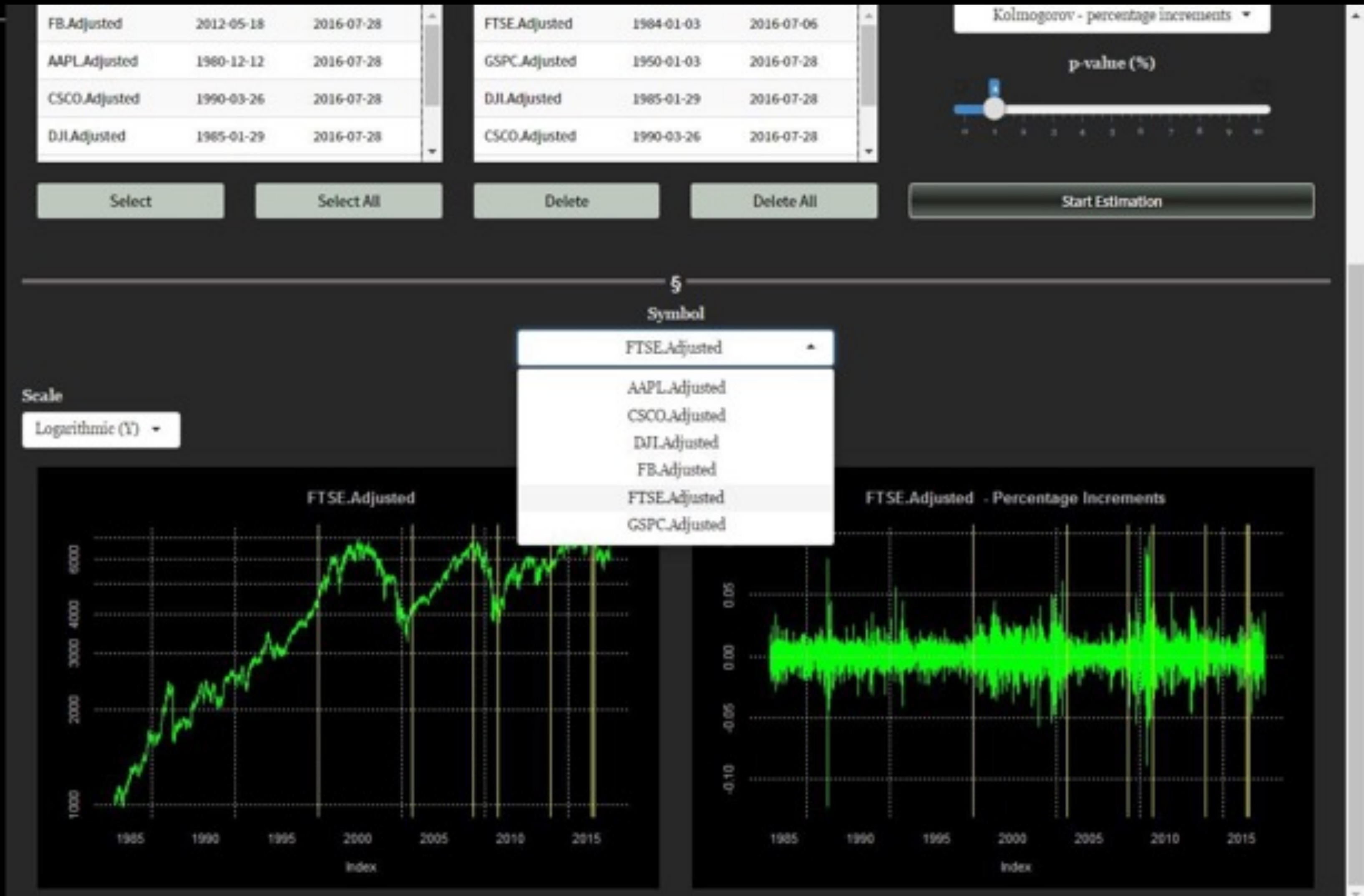
Loading Data → Explorative Data Analysis → Modeling → Simulation

Change Point Estimation



Loading Data → Explorative Data Analysis → Modeling → Simulation

Change Point Estimation

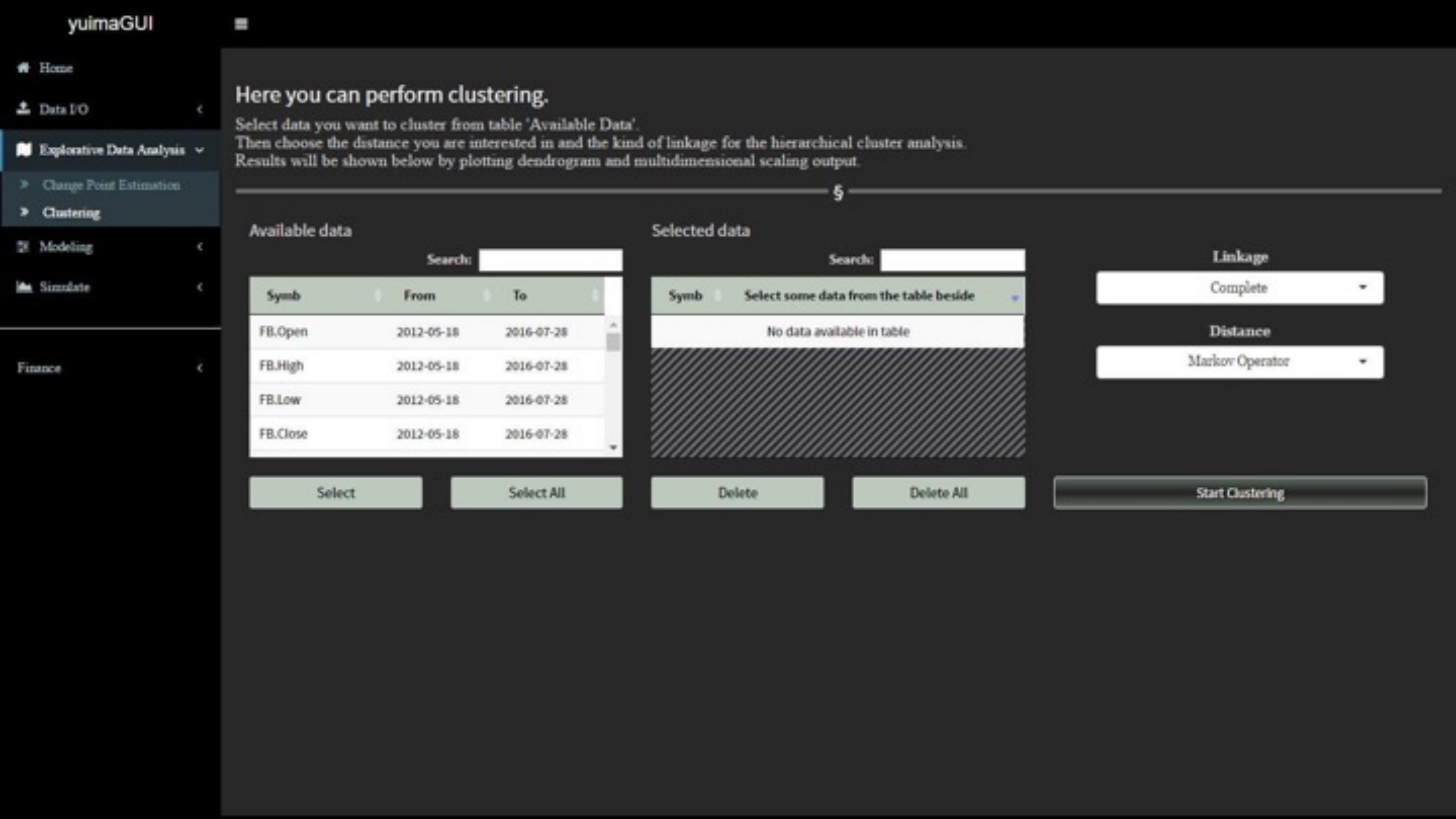


Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Change Point Estimation

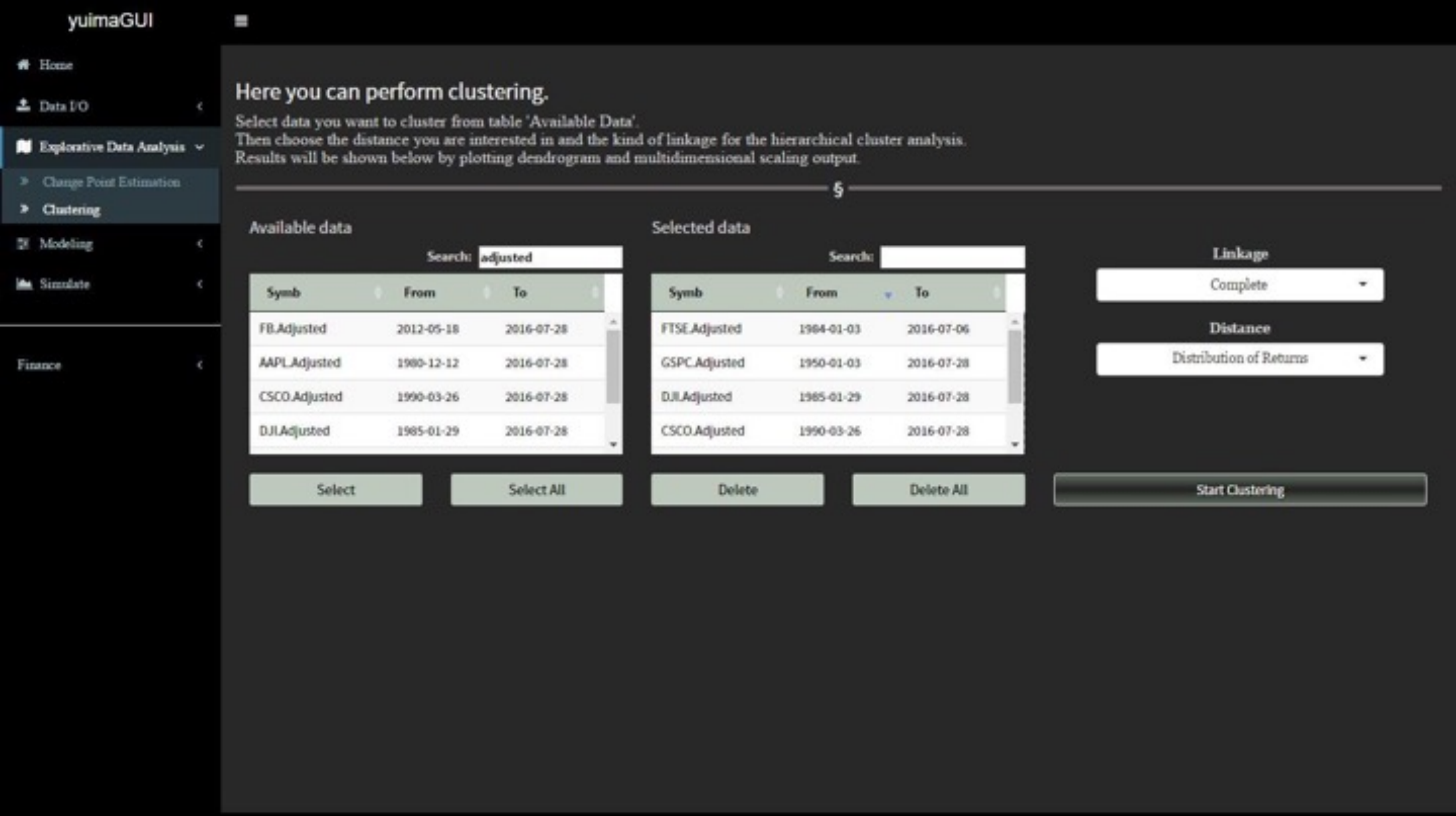
Loading Data → Explorative Data Analysis → Modeling → Simulation

Clustering



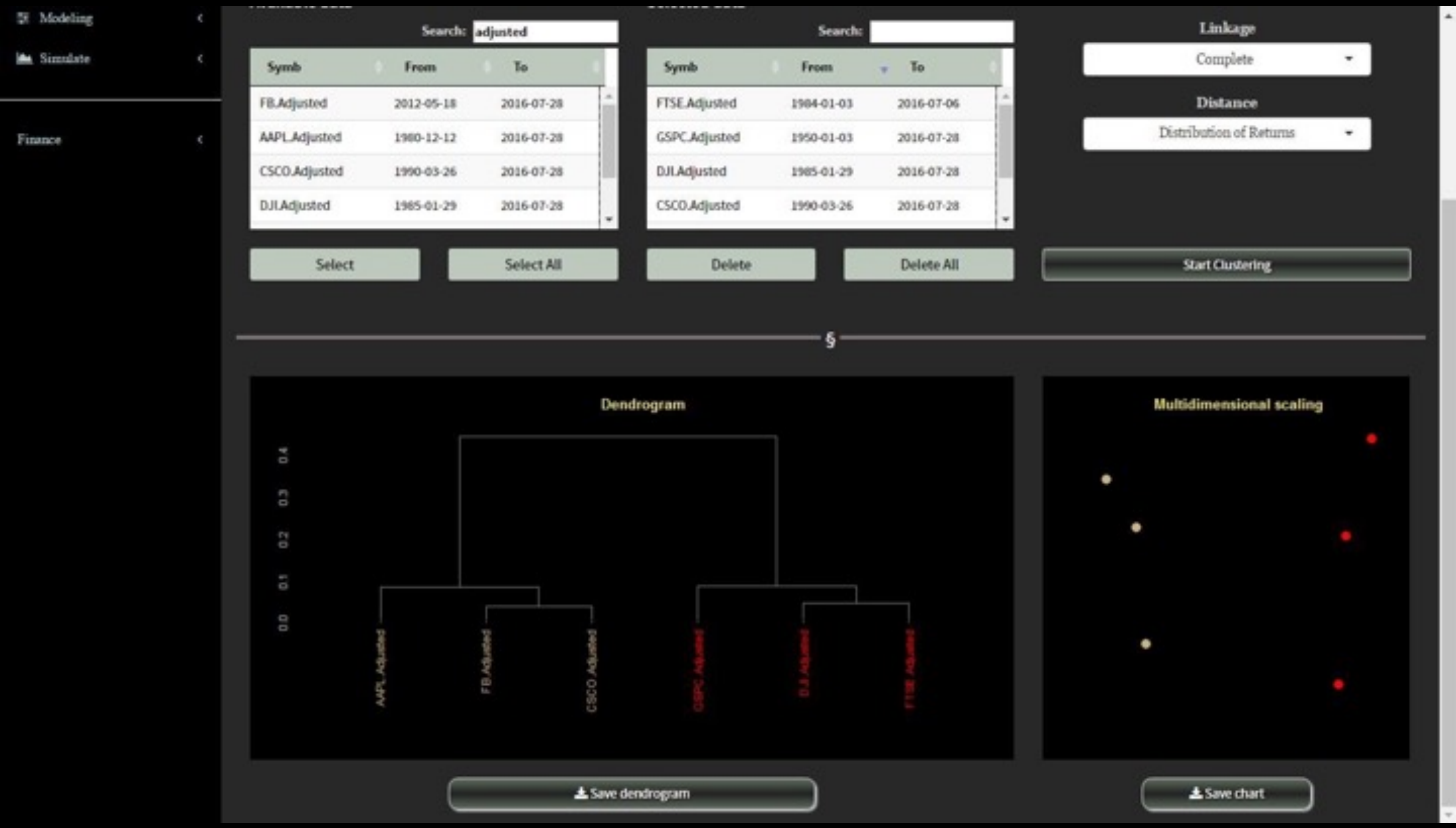
Loading Data → Explorative Data Analysis → Modeling → Simulation

Clustering



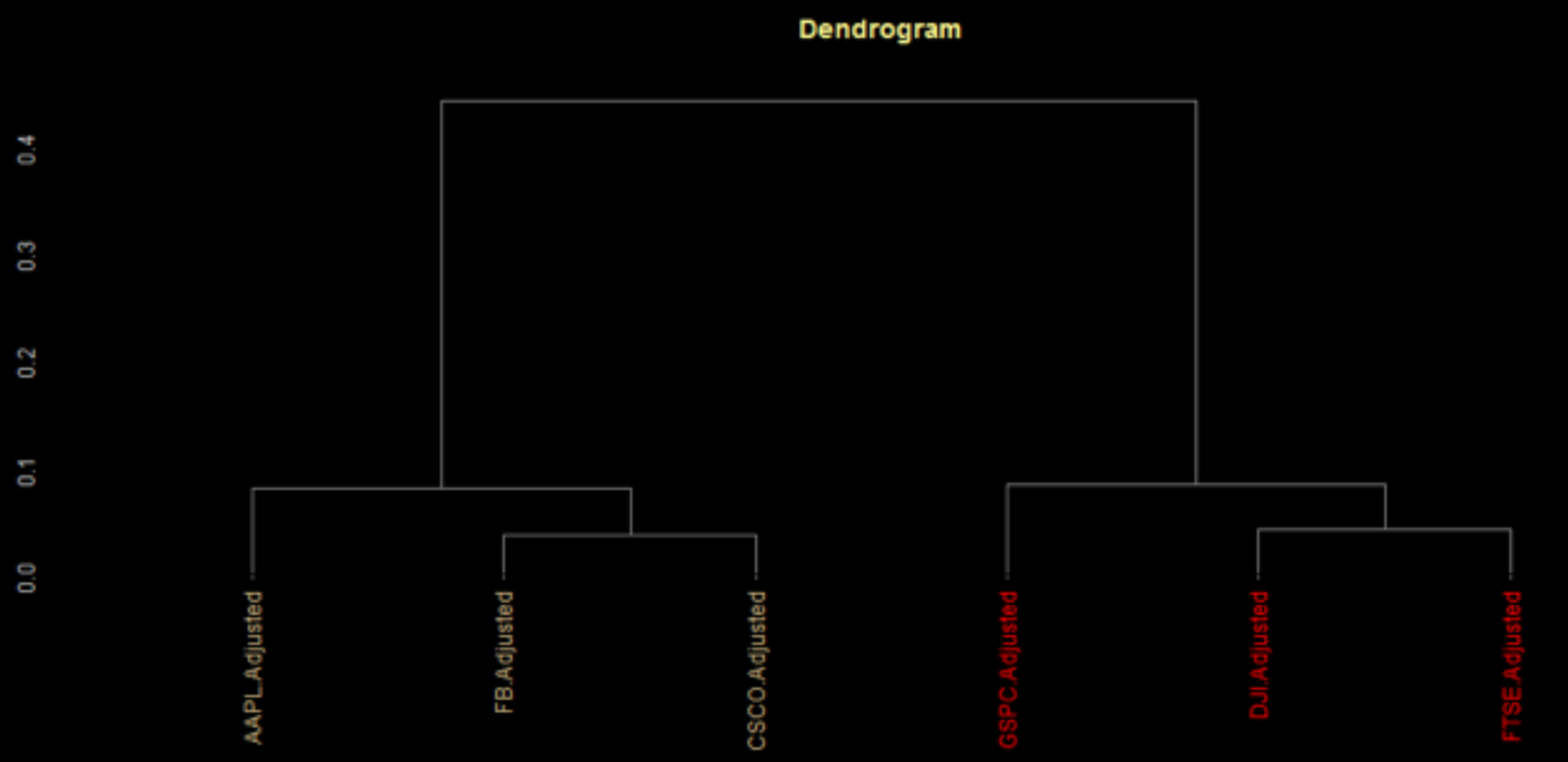
Loading Data → Explorative Data Analysis → Modeling → Simulation

Clustering



Loading Data → Explorative Data Analysis → Modeling → Simulation

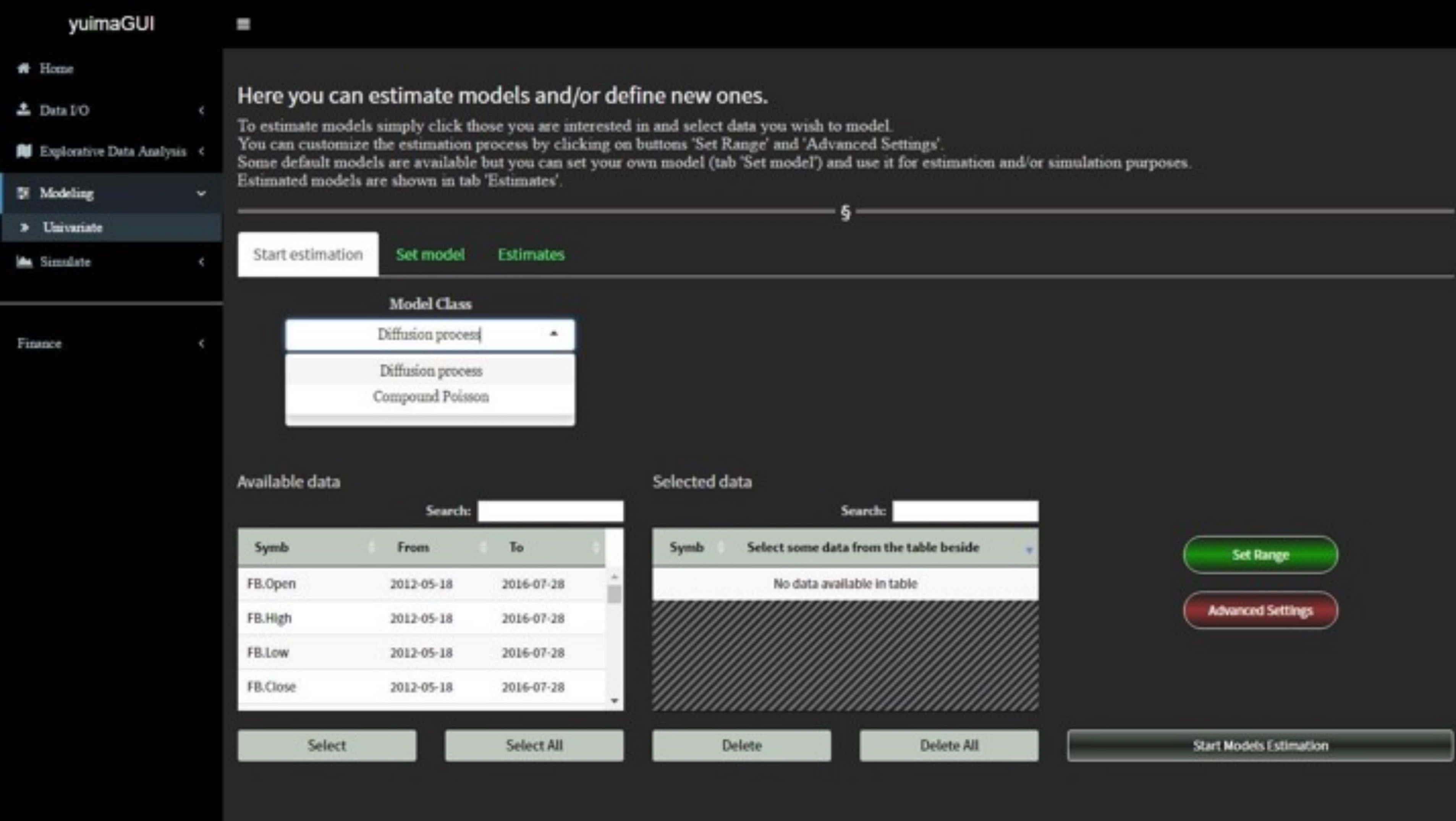
Clustering



Loading Data → Explorative Data Analysis → Modeling → Simulation

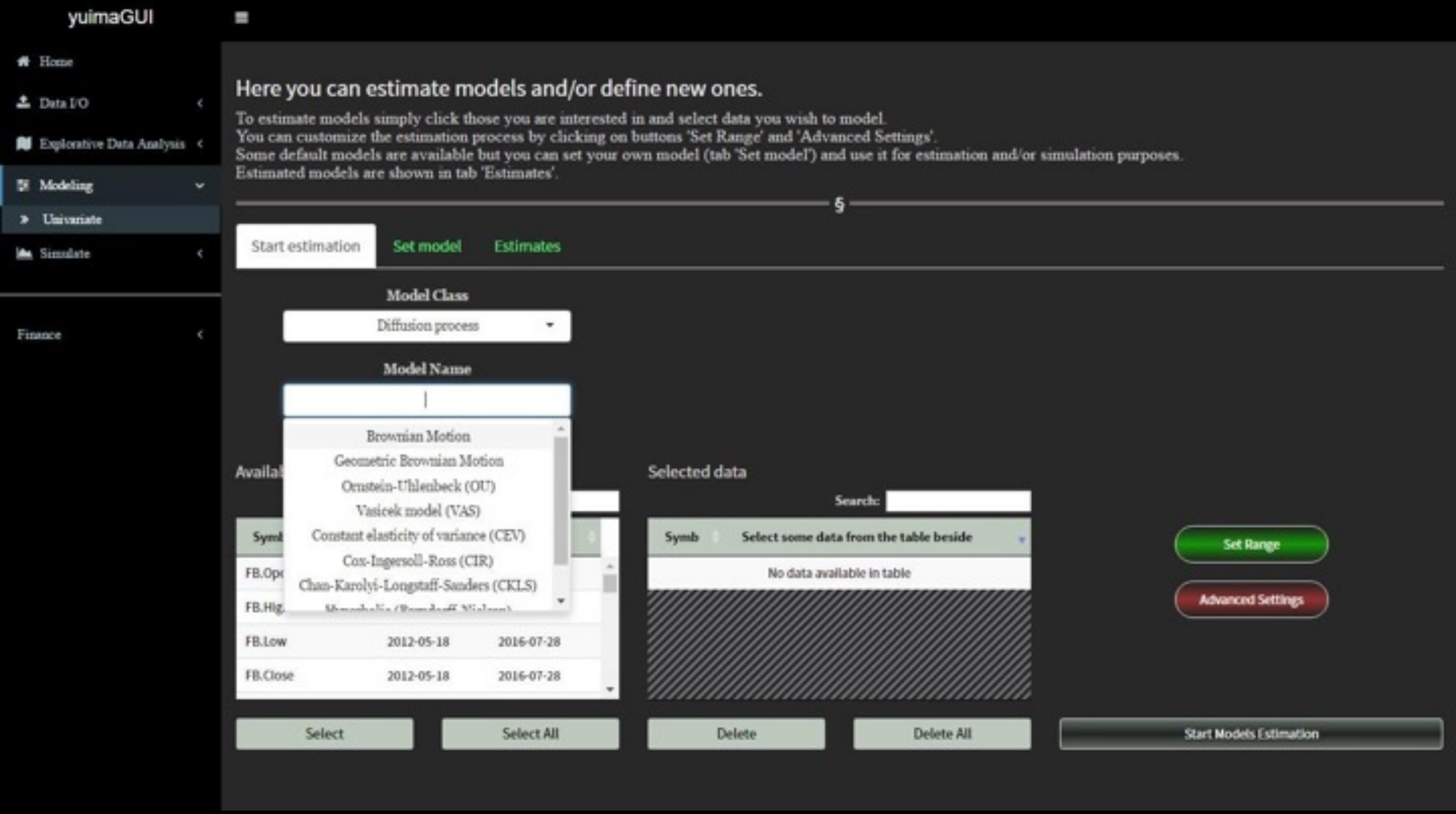
Loading Data → Explorative Data Analysis → Modeling → Simulation

Univariate Modeling



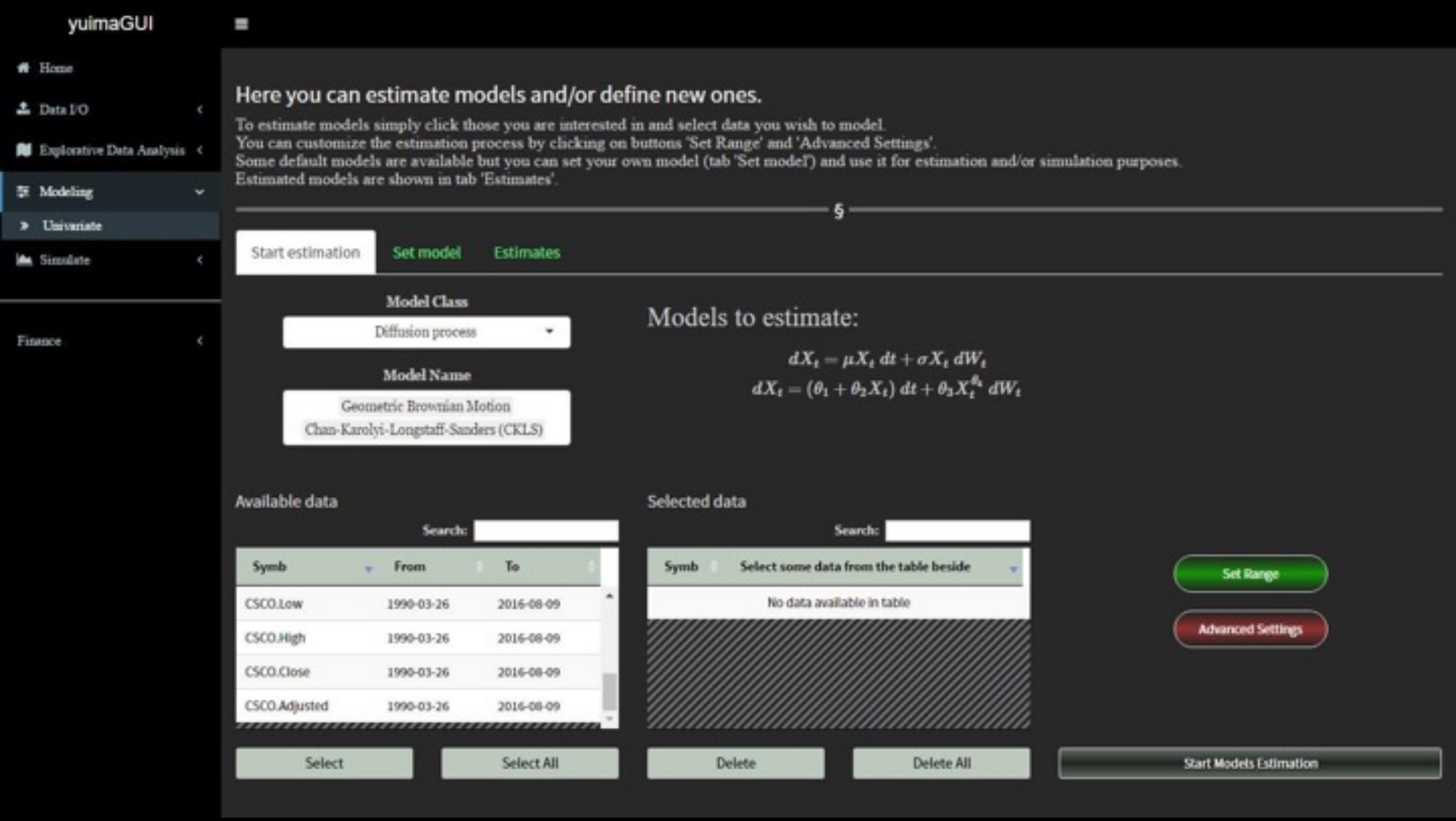
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Univariate Modeling



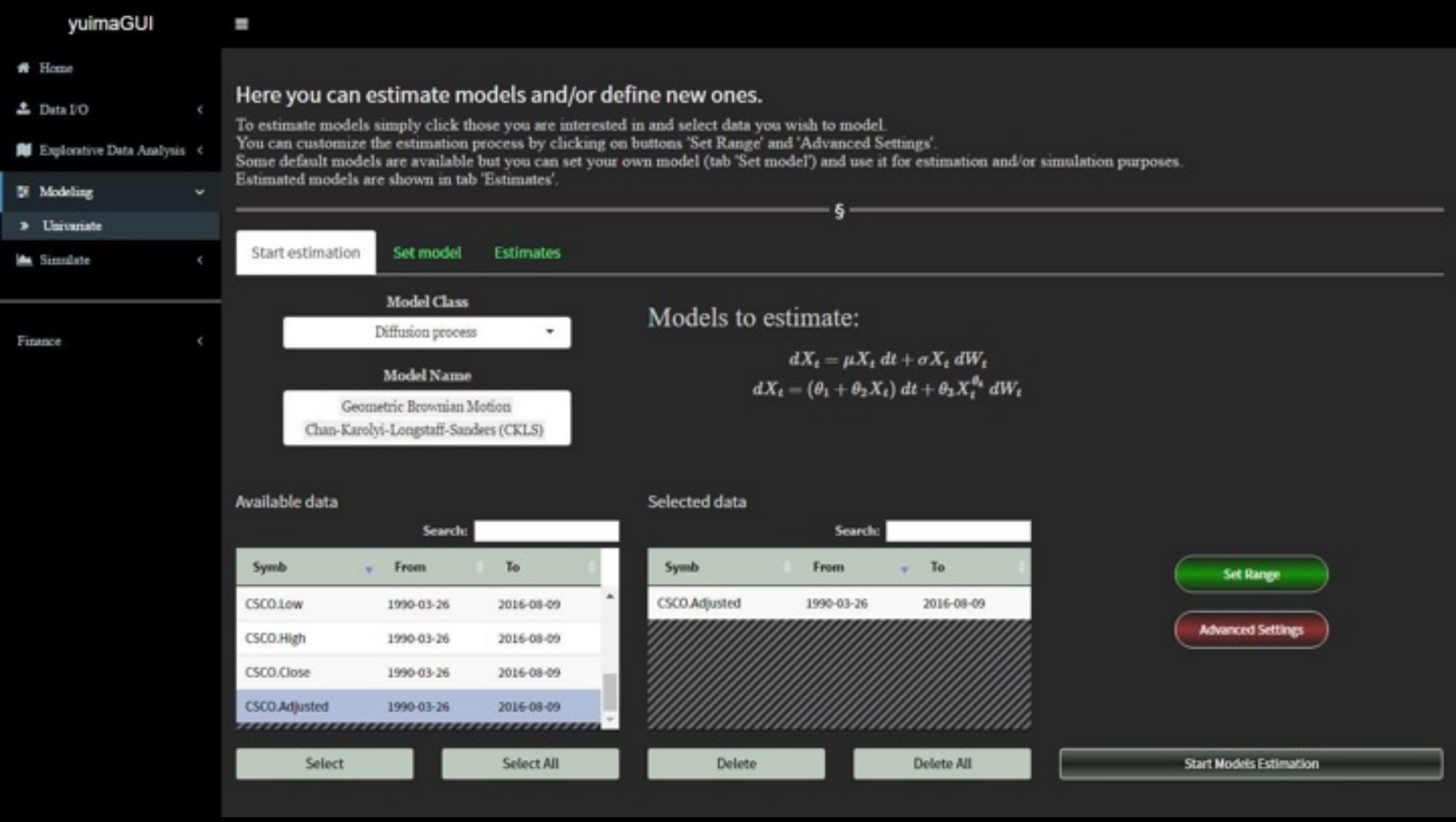
Loading Data → Explorative Data Analysis → Modeling → Simulation

Univariate Modeling



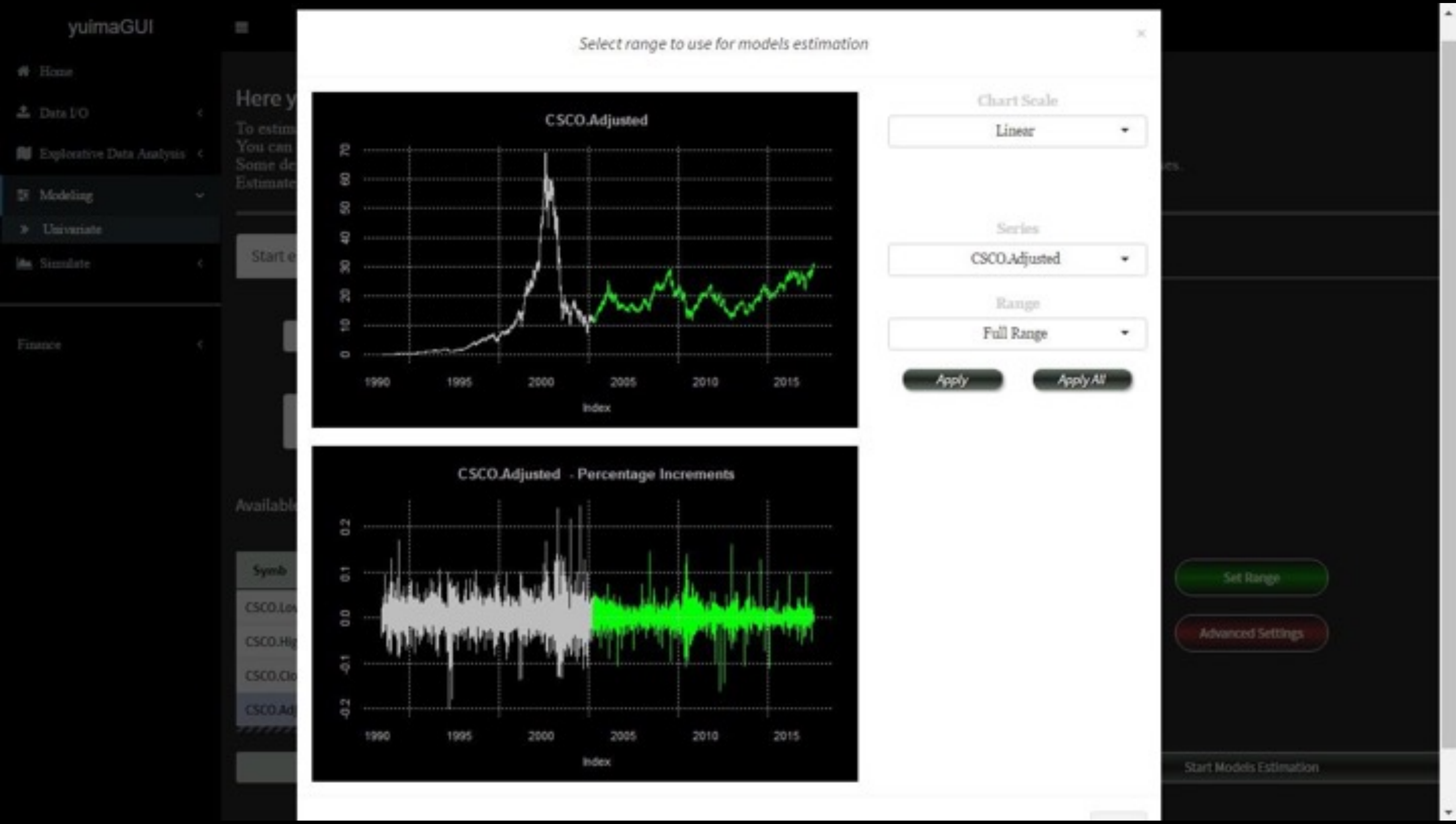
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Univariate Modeling



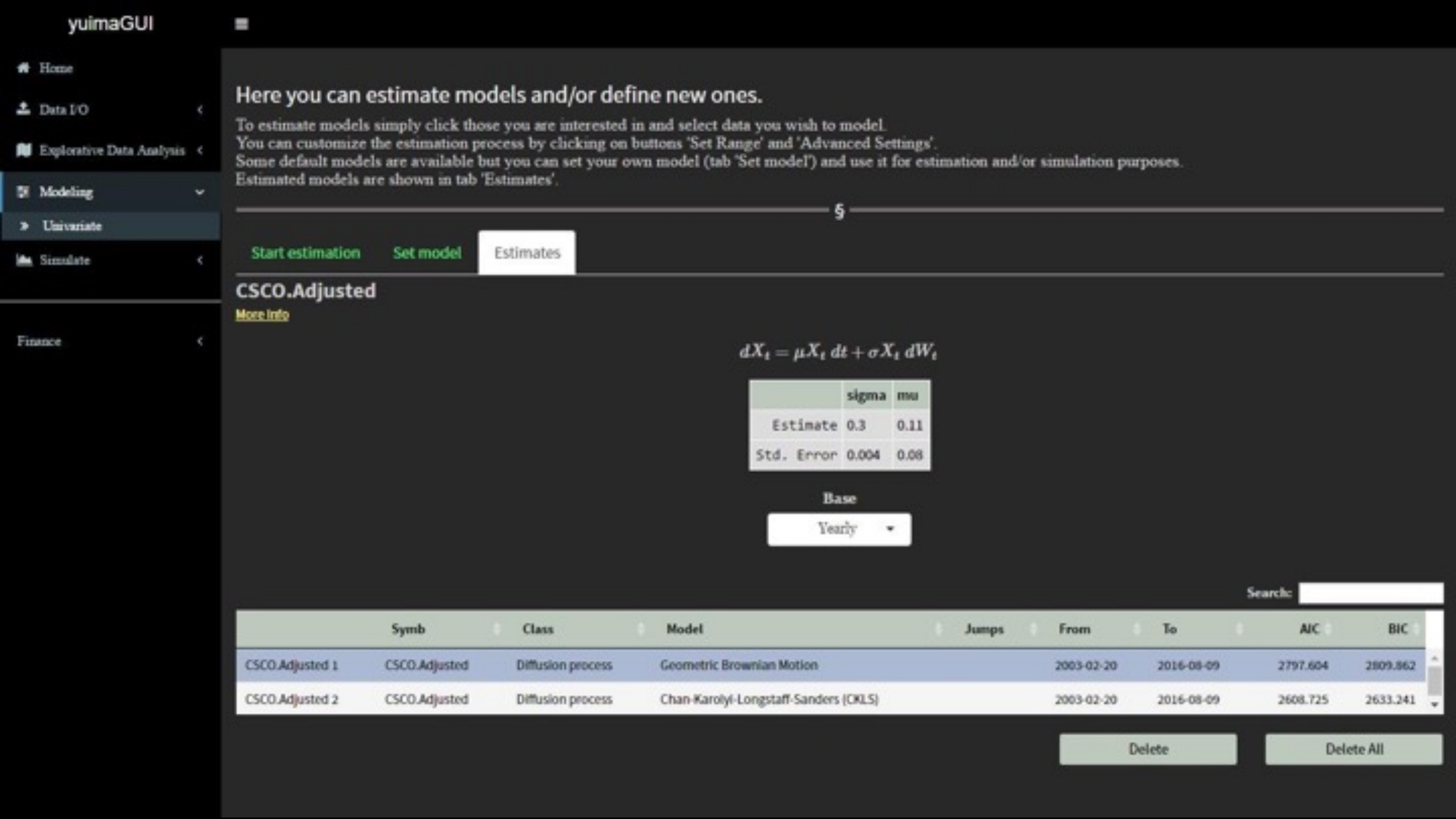
Loading Data → Explorative Data Analysis → Modeling → Simulation

Univariate Modeling



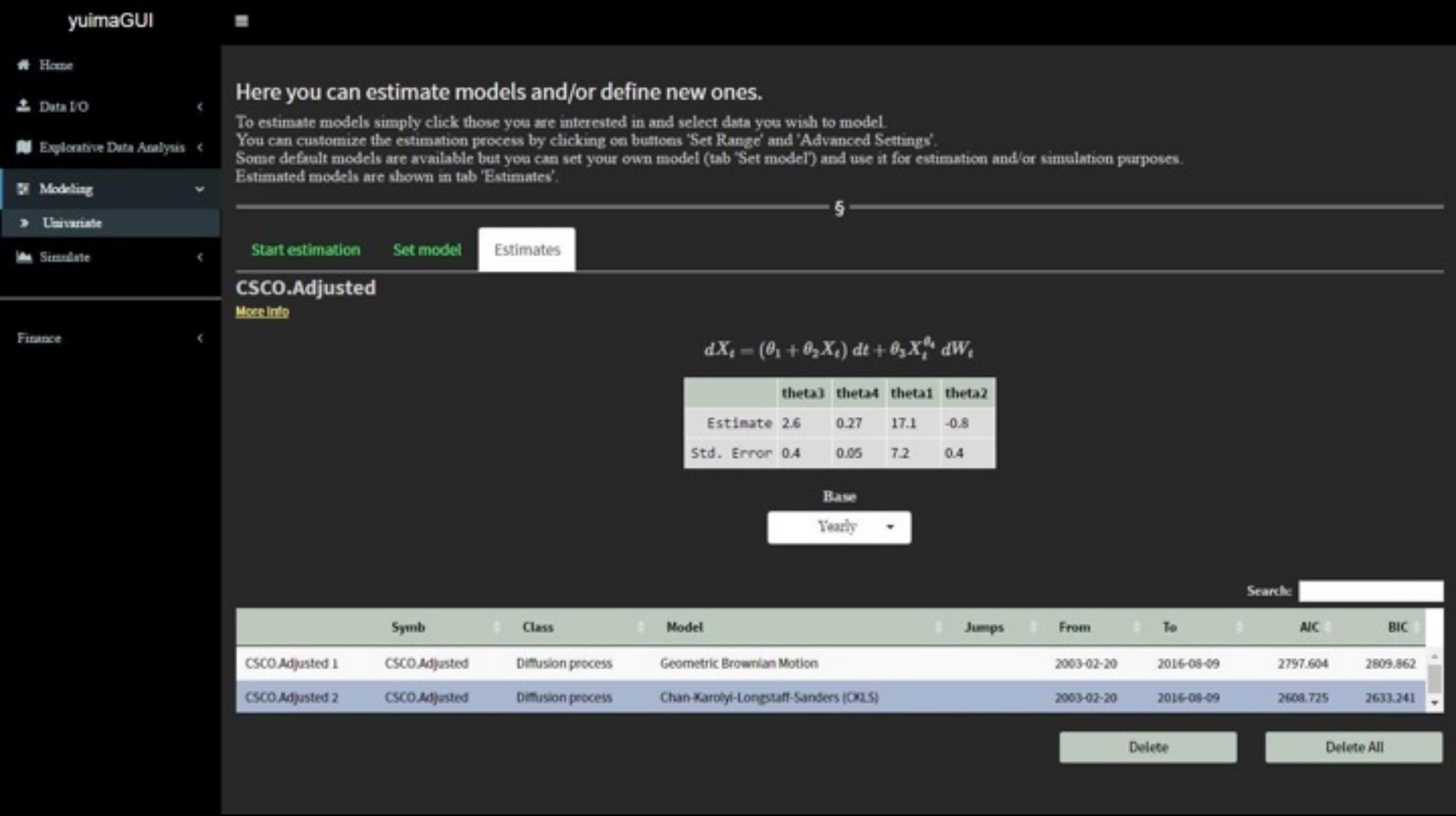
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Univariate Modeling



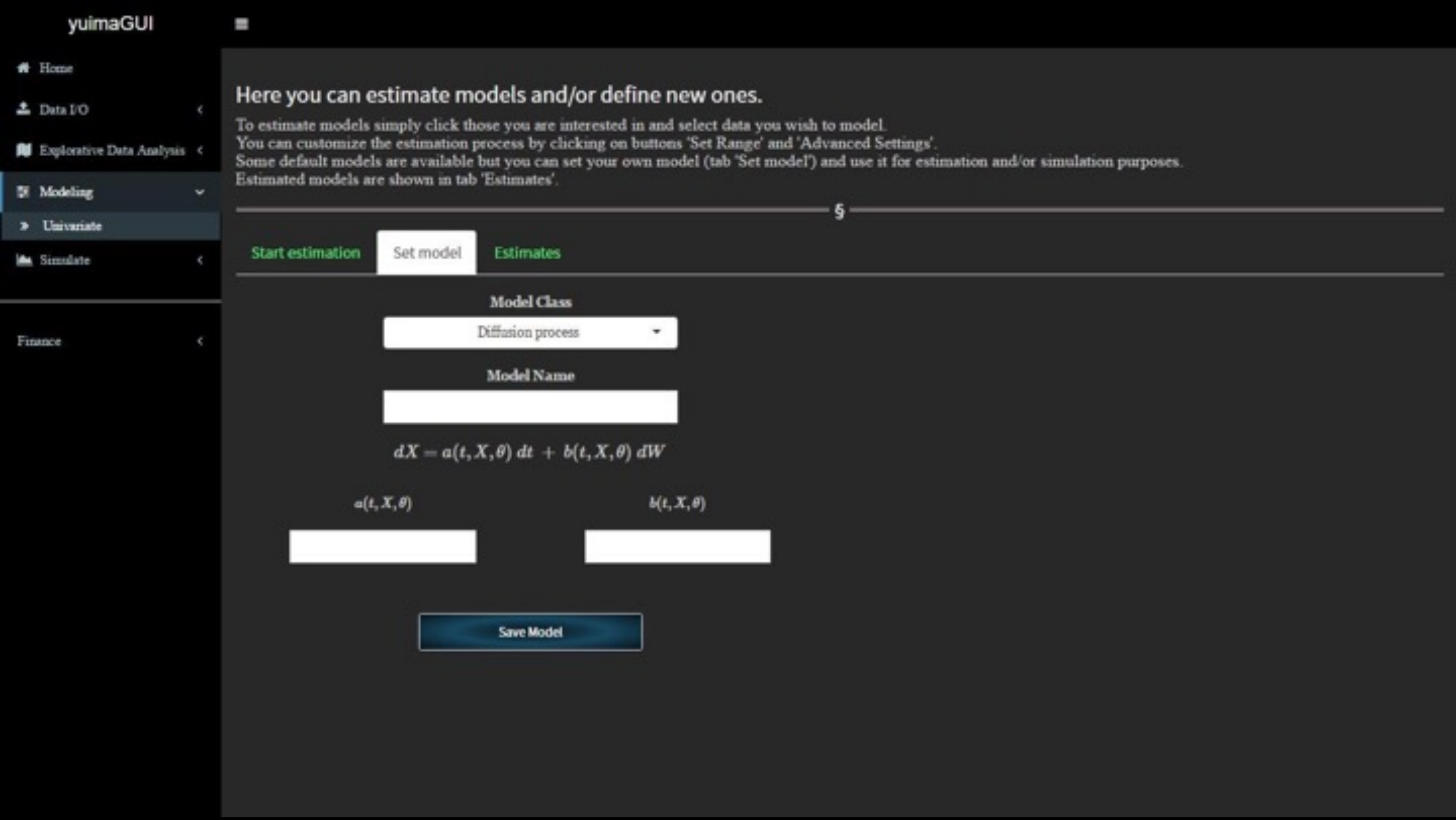
Loading Data → Explorative Data Analysis → Modeling → Simulation

Univariate Modeling



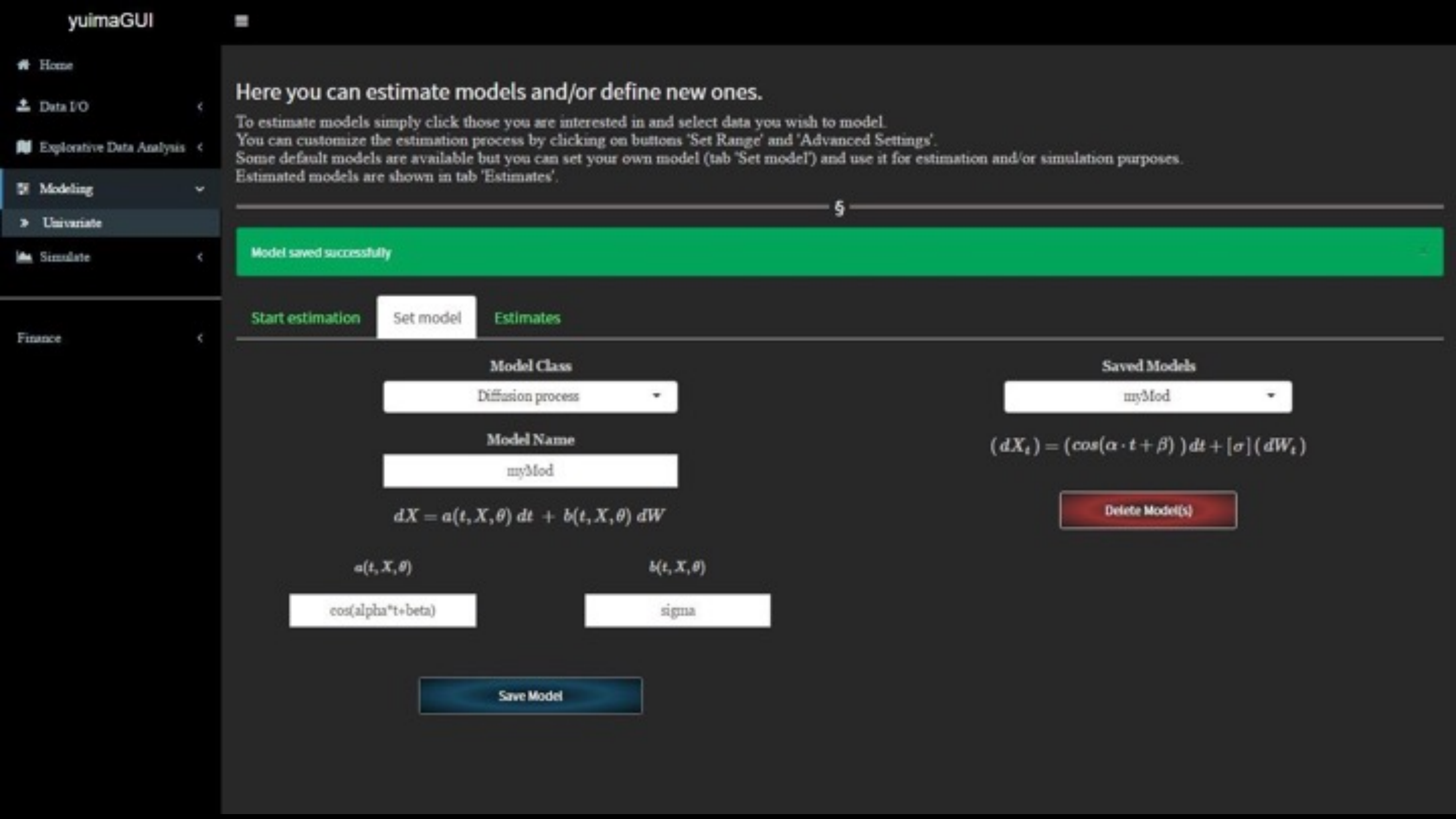
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Building your model



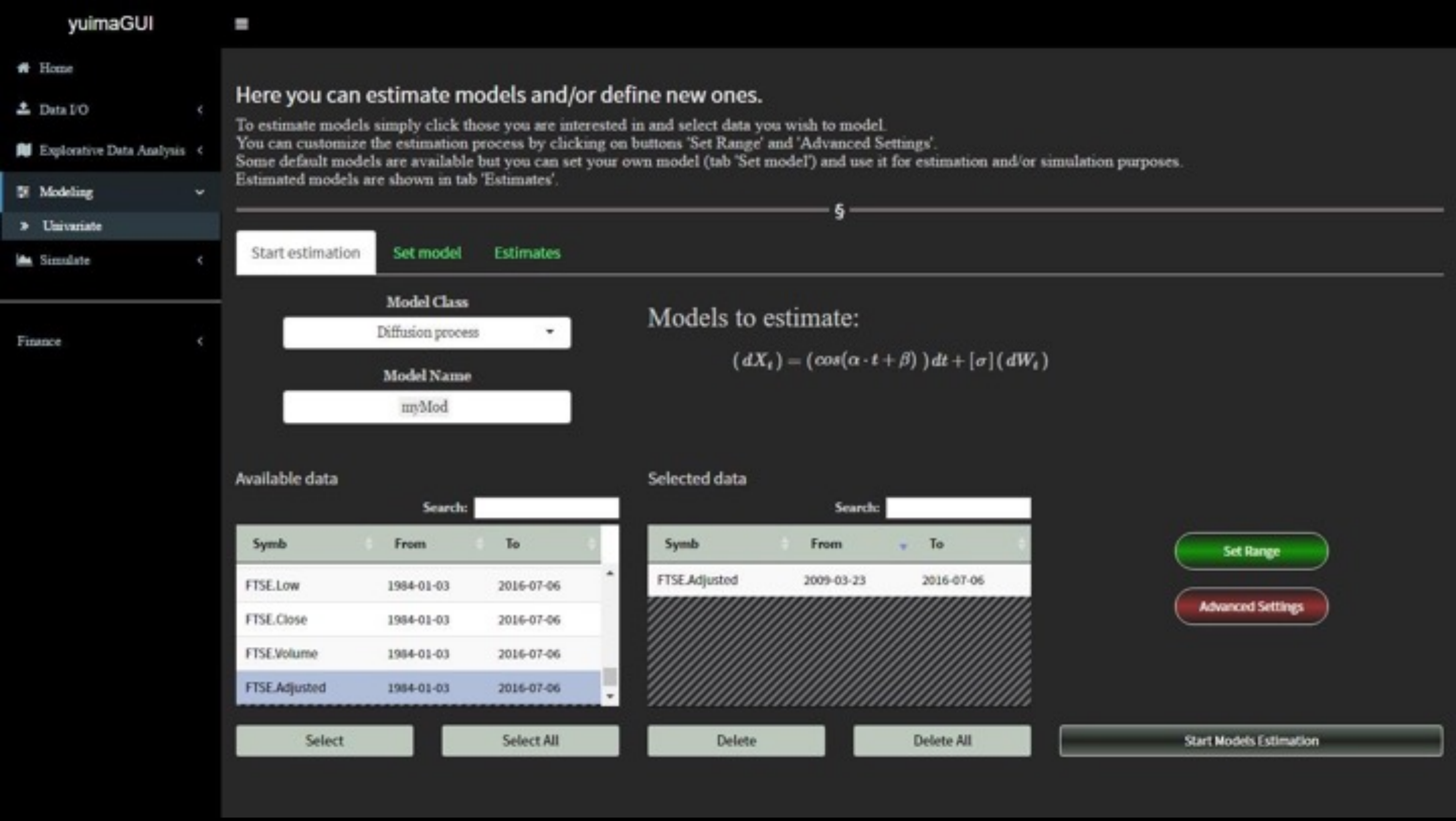
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Building your model



Loading Data → Explorative Data Analysis → Modeling → Simulation

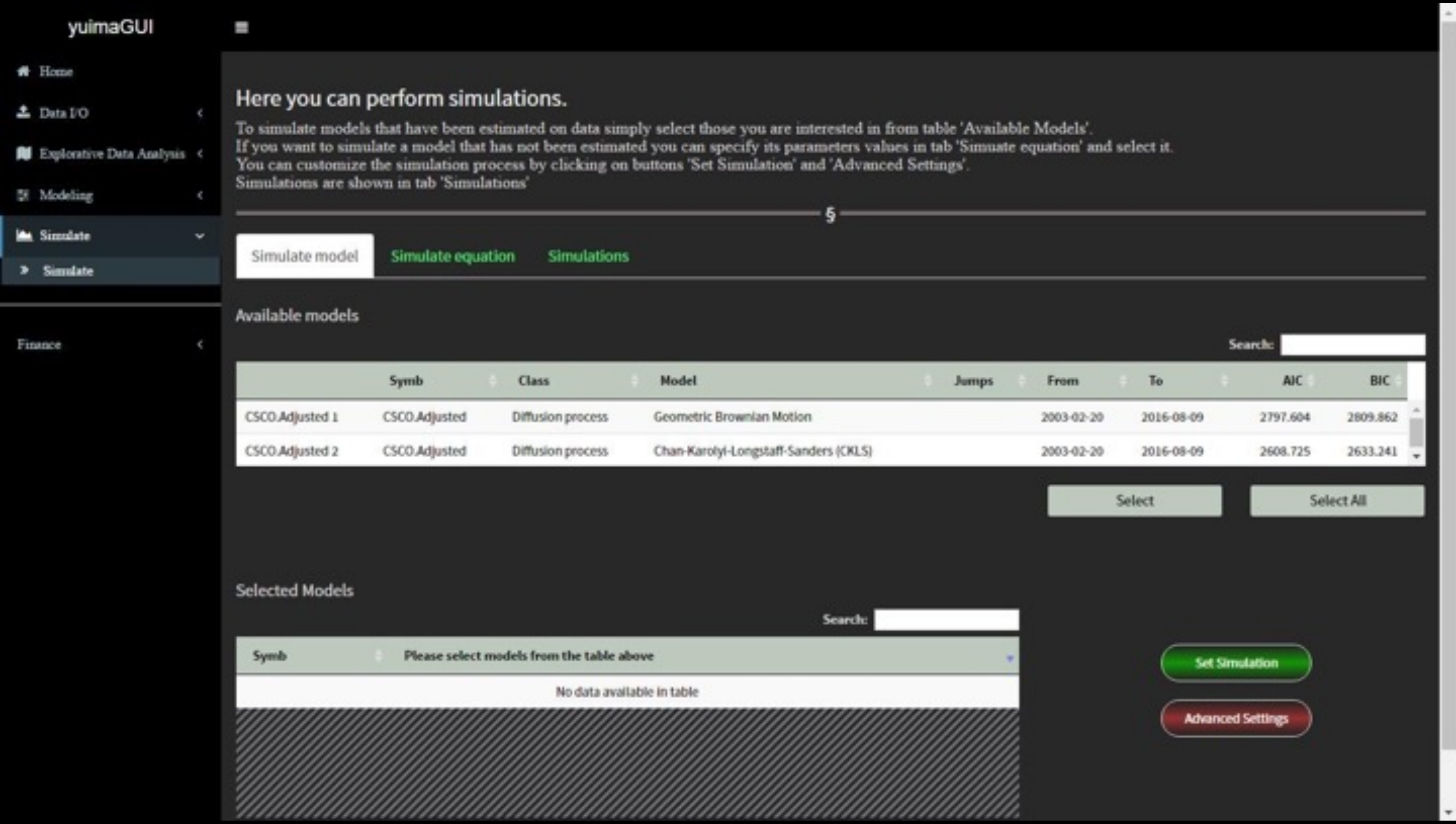
Building your model



Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

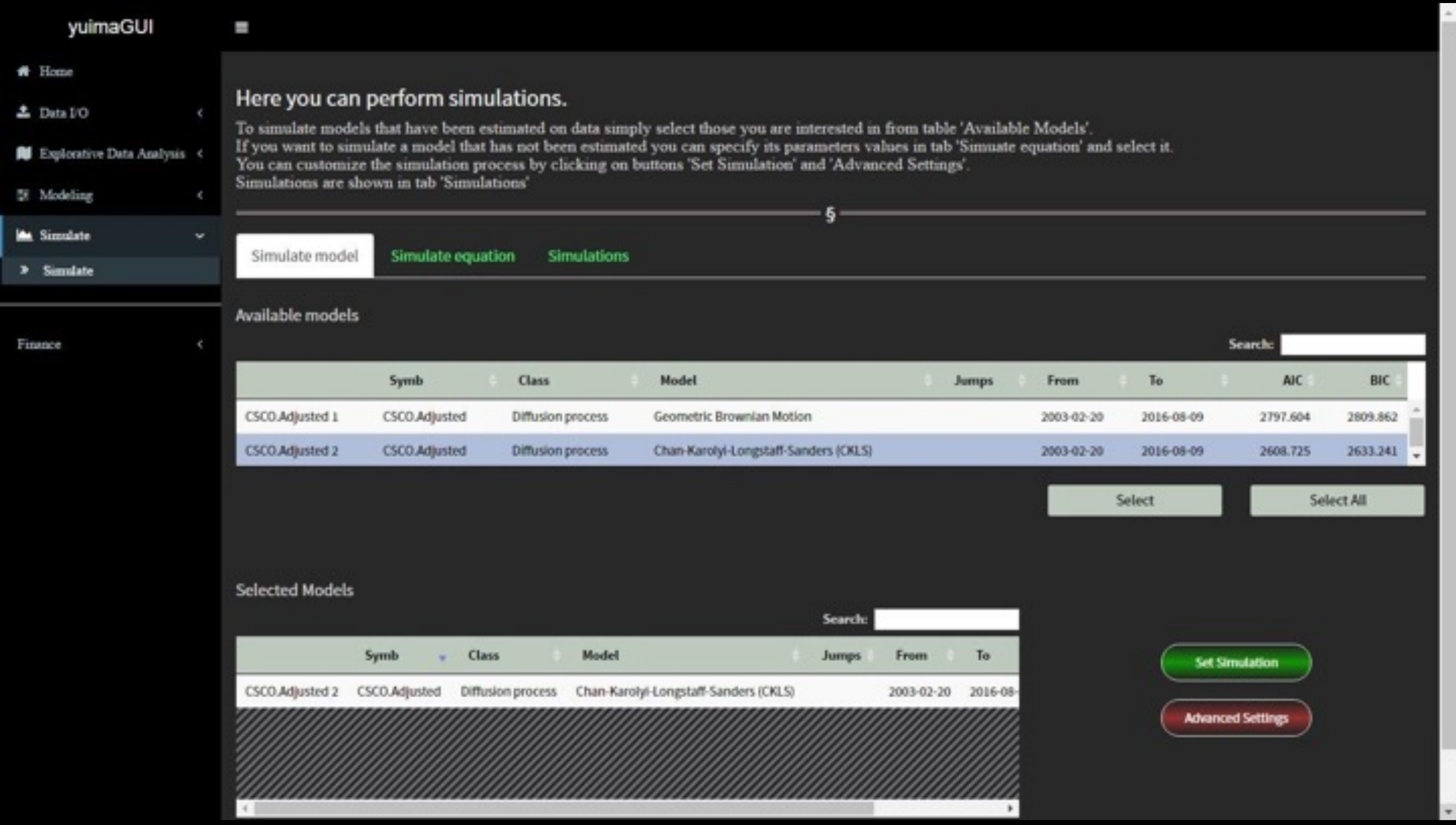
Loading Data → Explorative Data Analysis → Modeling → Simulation

Simulate Estimated Model



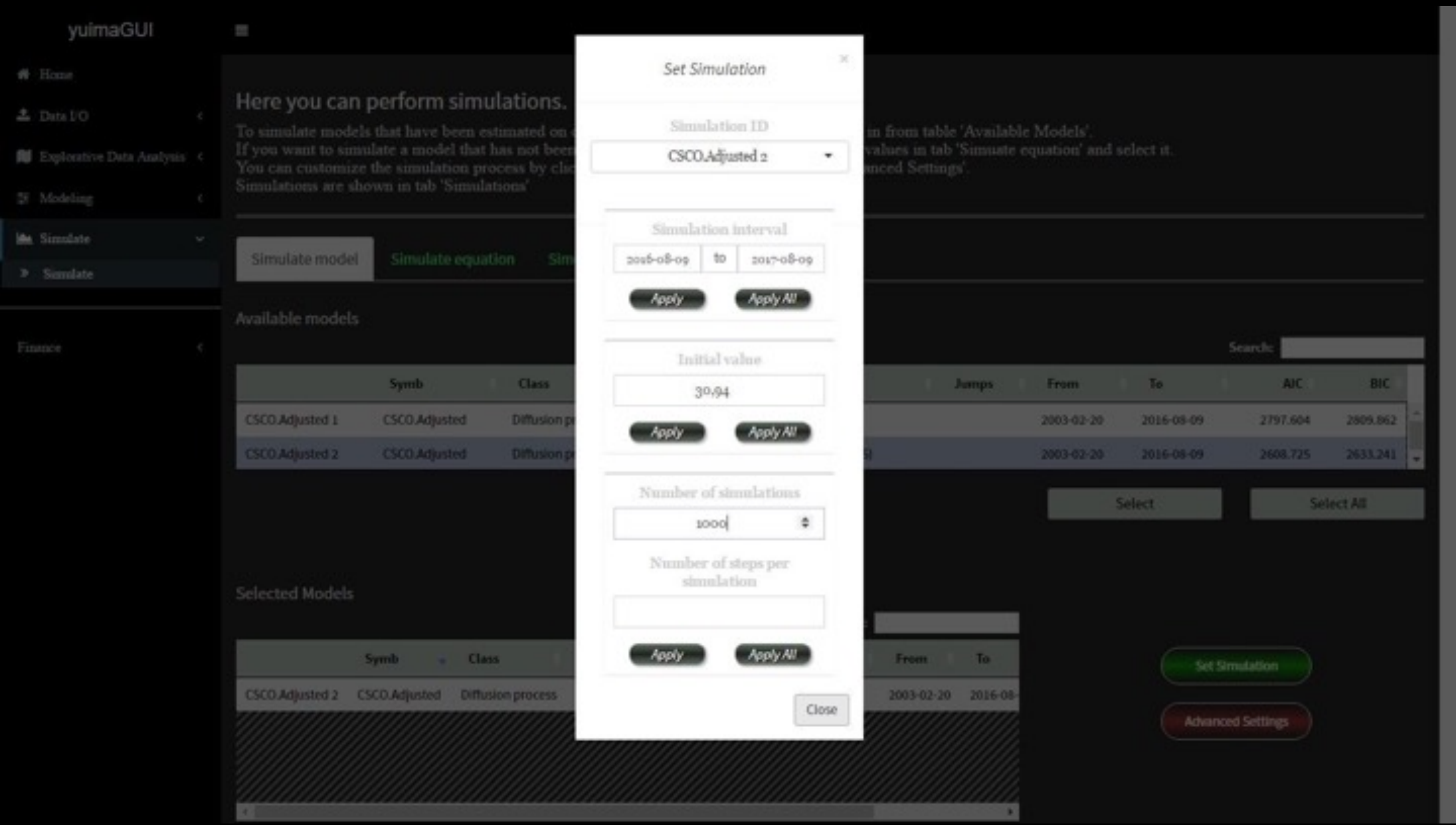
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Simulate Estimated Model



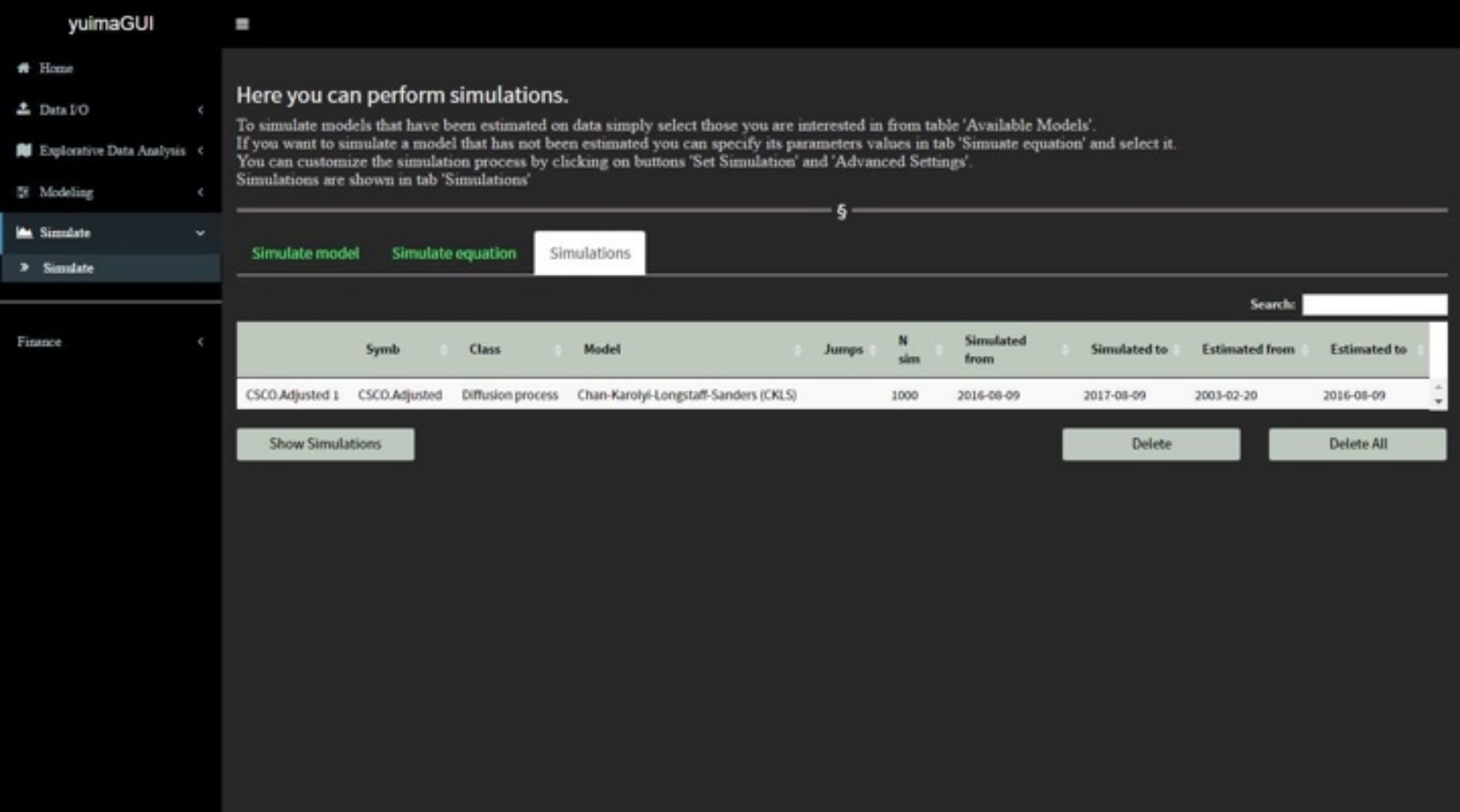
Loading Data → Explorative Data Analysis → Modeling → Simulation

Simulate Estimated Model



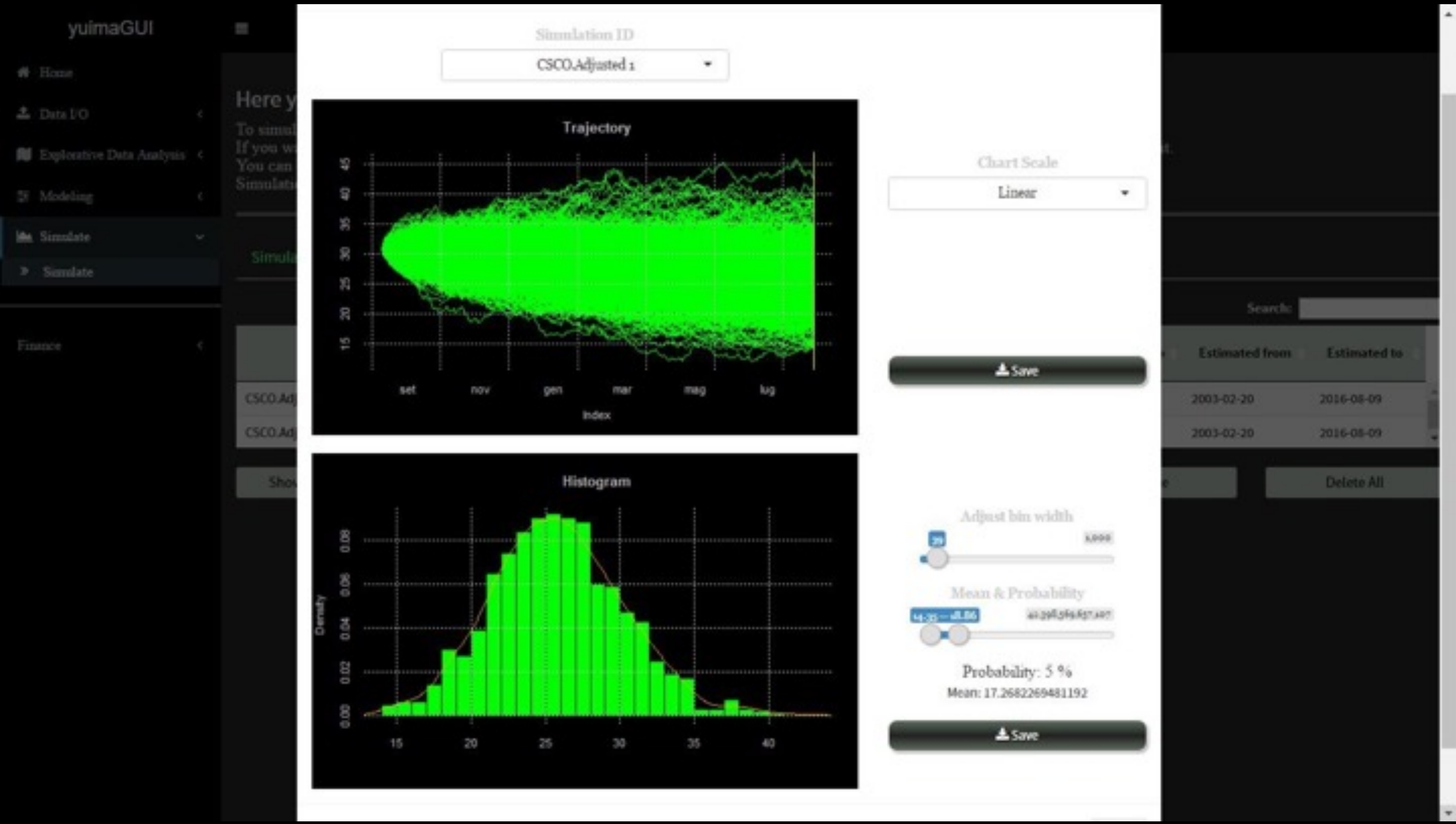
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Simulate Estimated Model



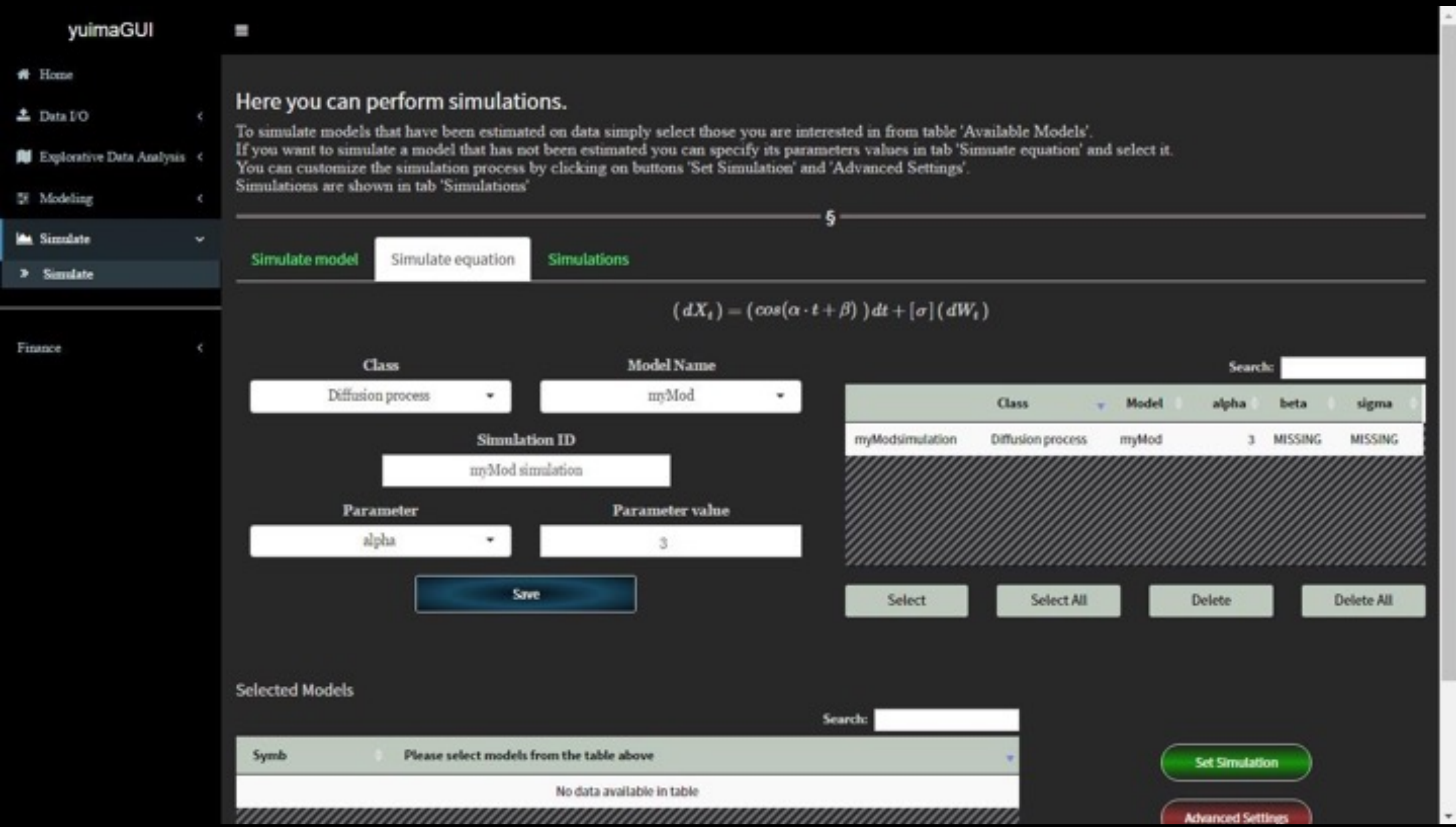
Loading Data → Explorative Data Analysis → Modeling → Simulation

Simulate Estimated Model



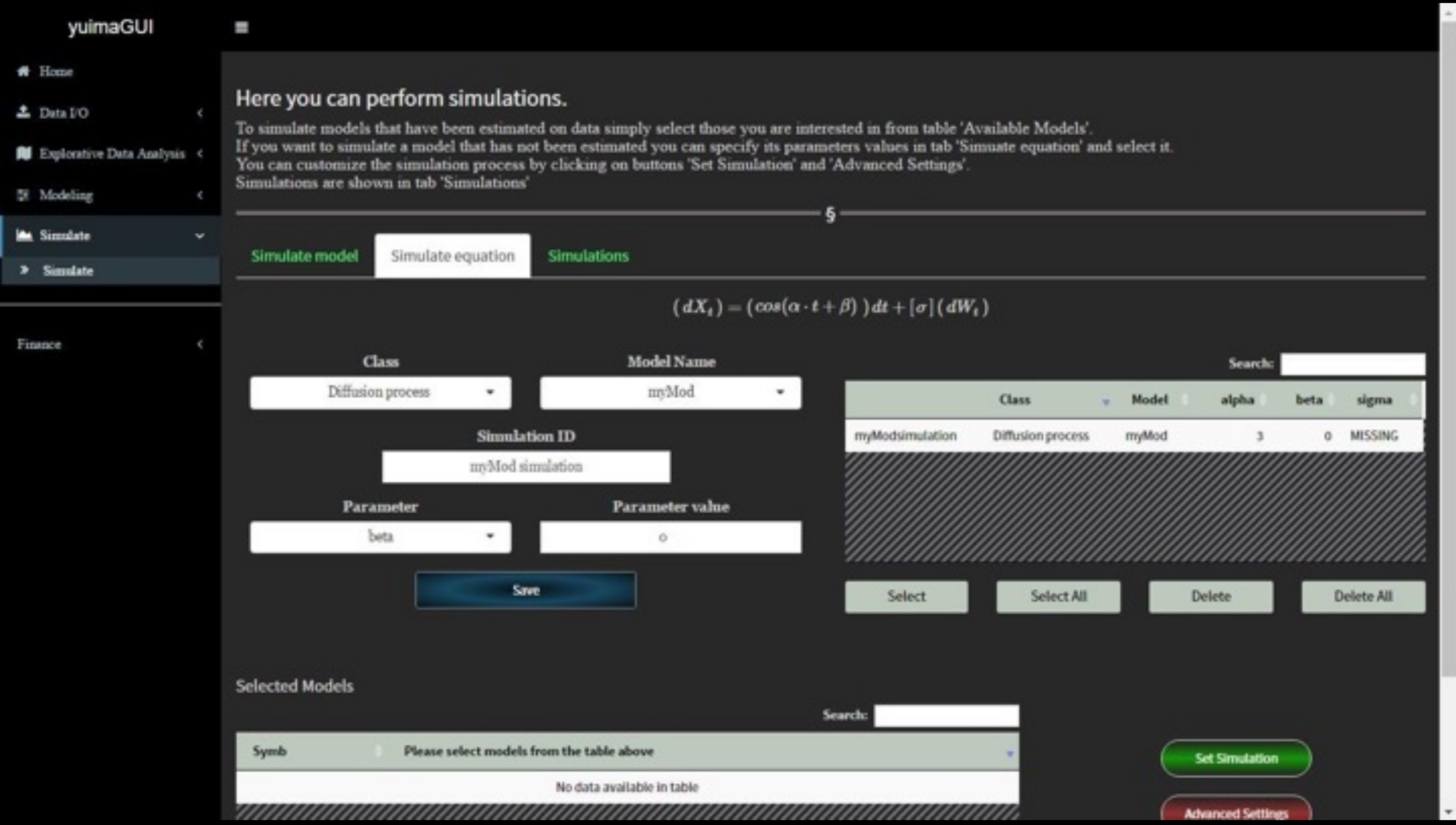
Loading Data → Explorative Data Analysis → Modeling → Simulation

Simulate Equation



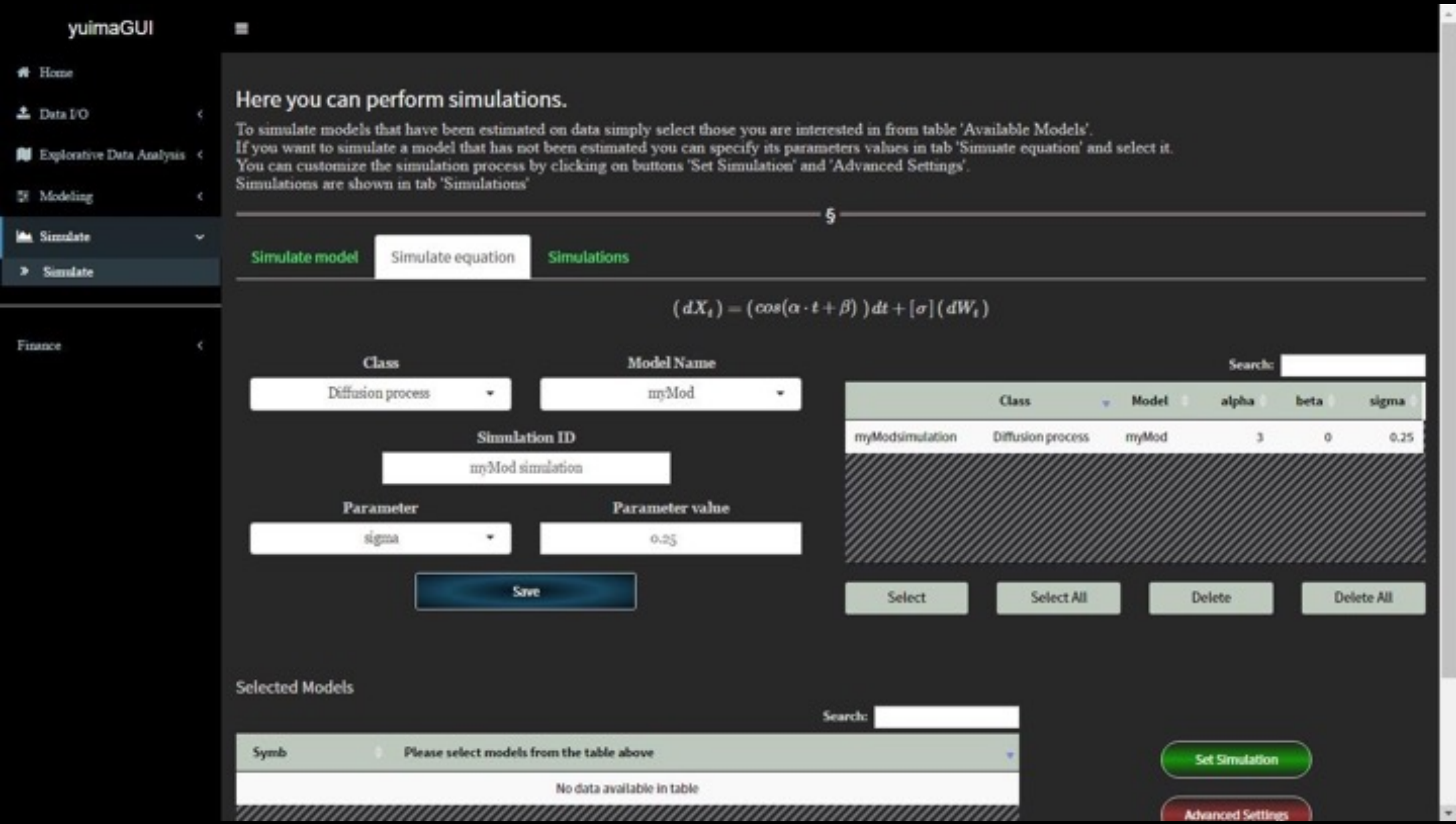
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Simulate Equation



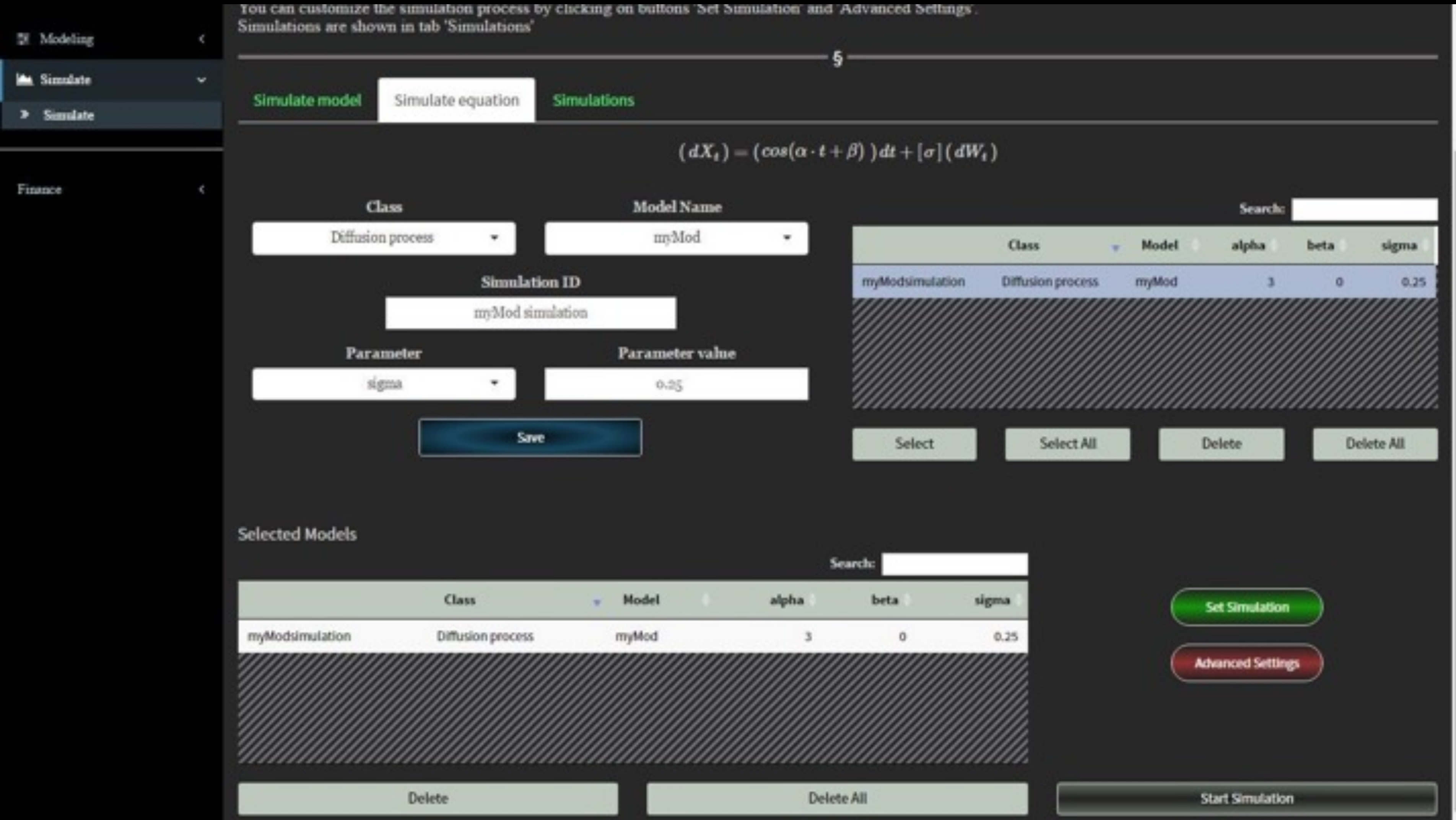
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Simulate Equation



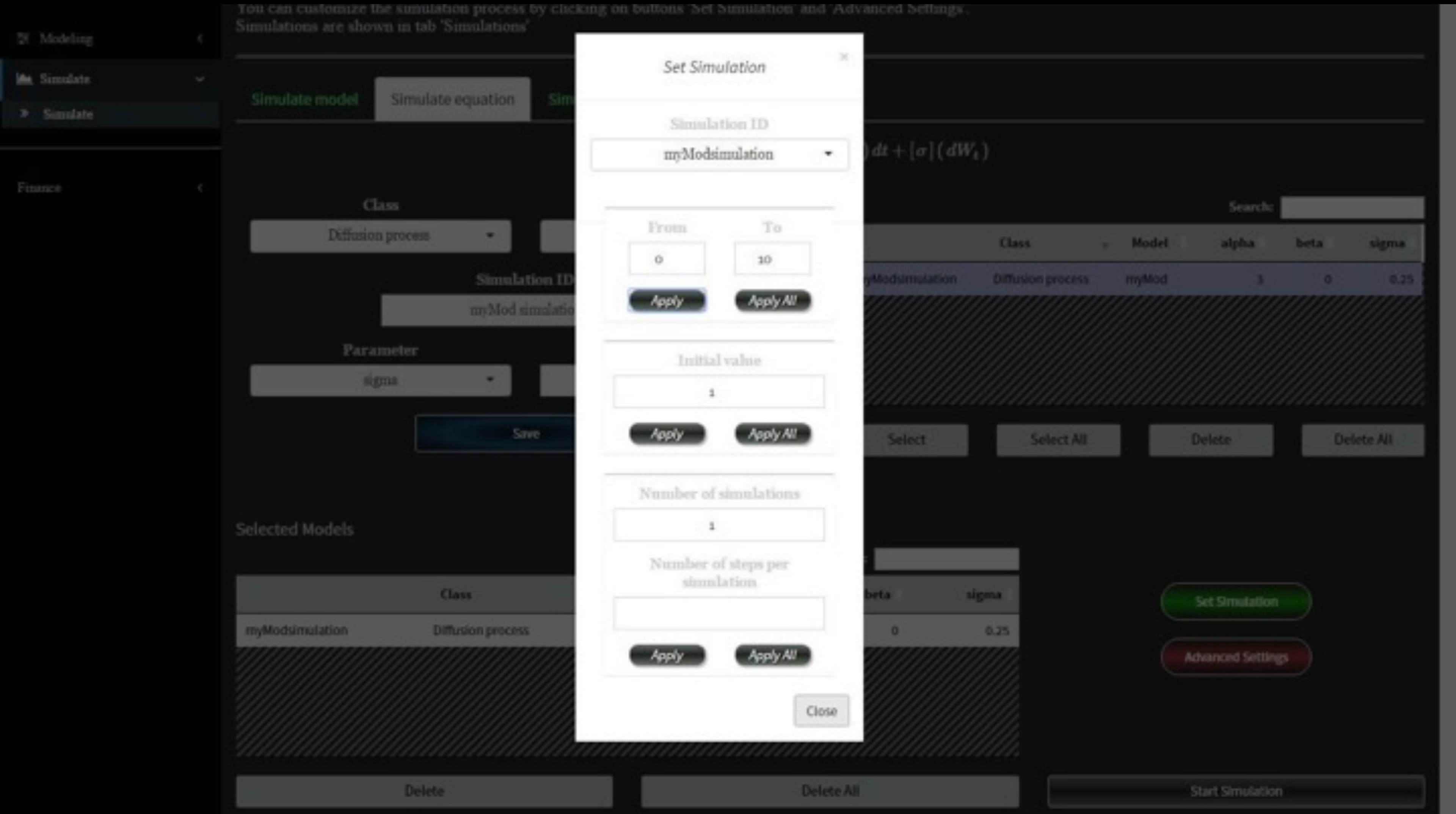
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Simulate Equation



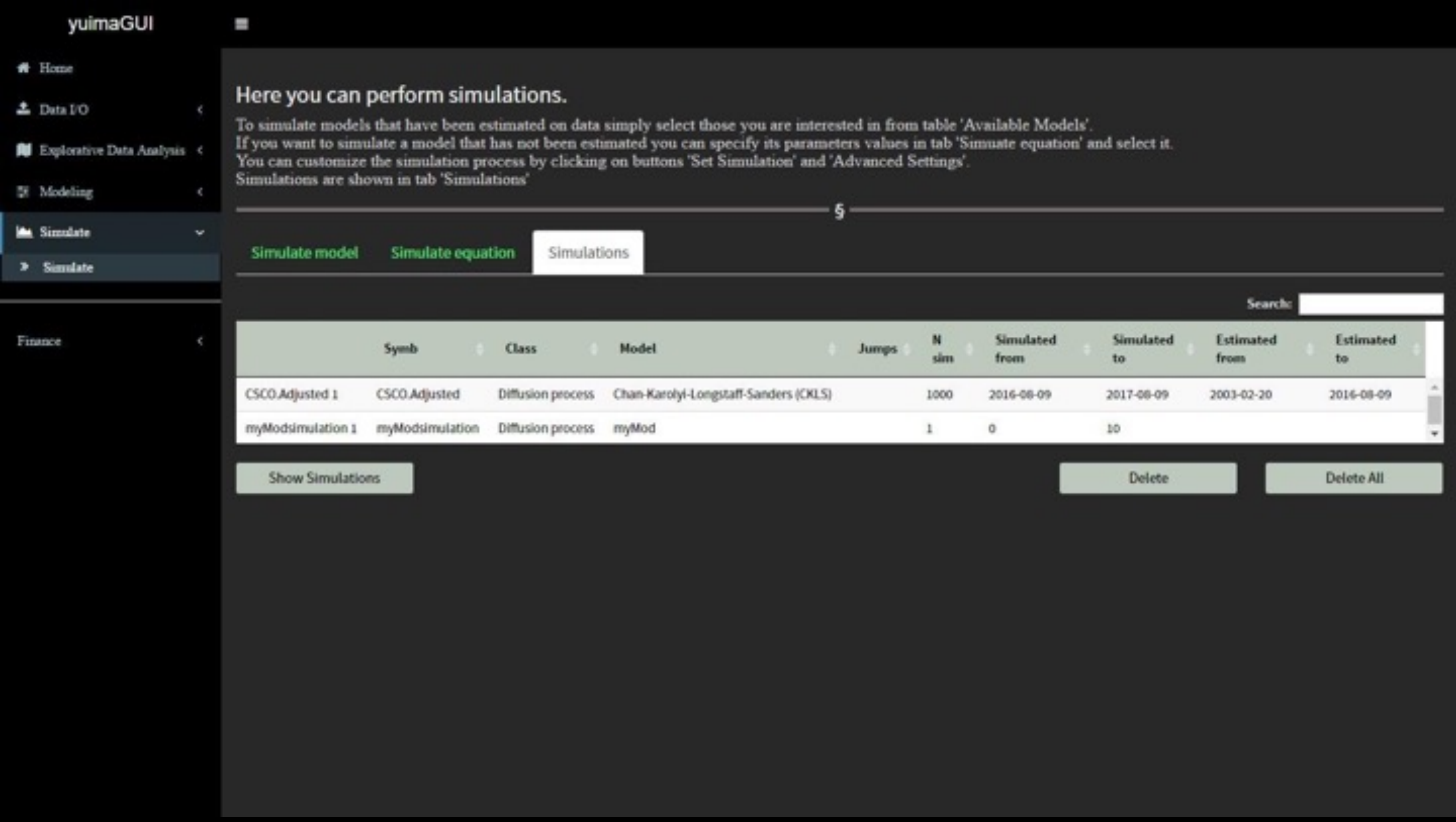
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Simulate Equation



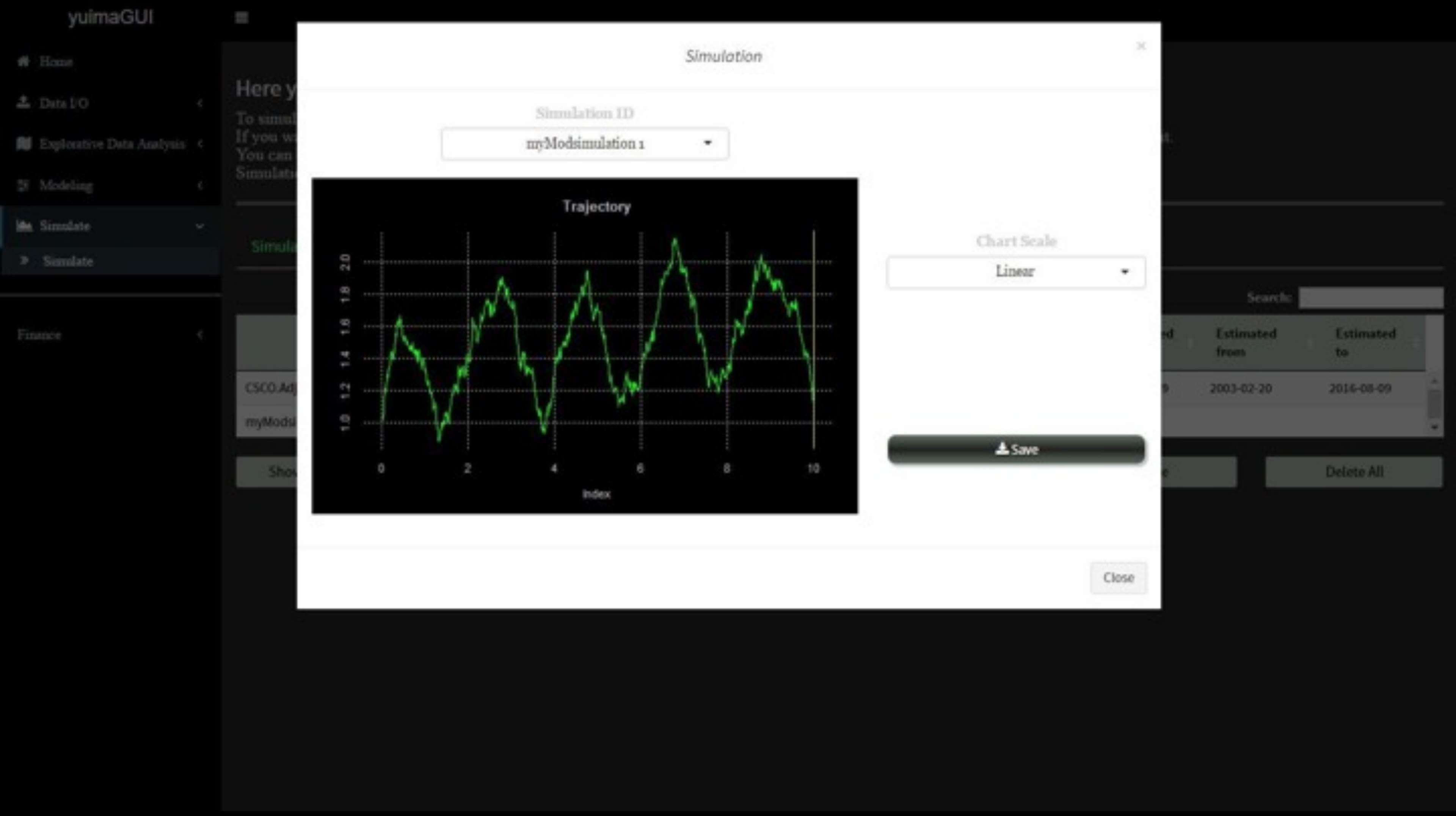
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Simulate Equation



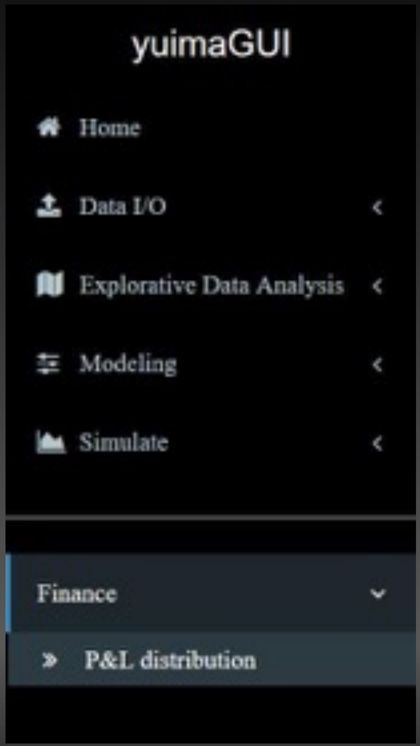
Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

Simulate Equation



Loading Data ⟶ Explorative Data Analysis ⟶ Modeling ⟶ Simulation

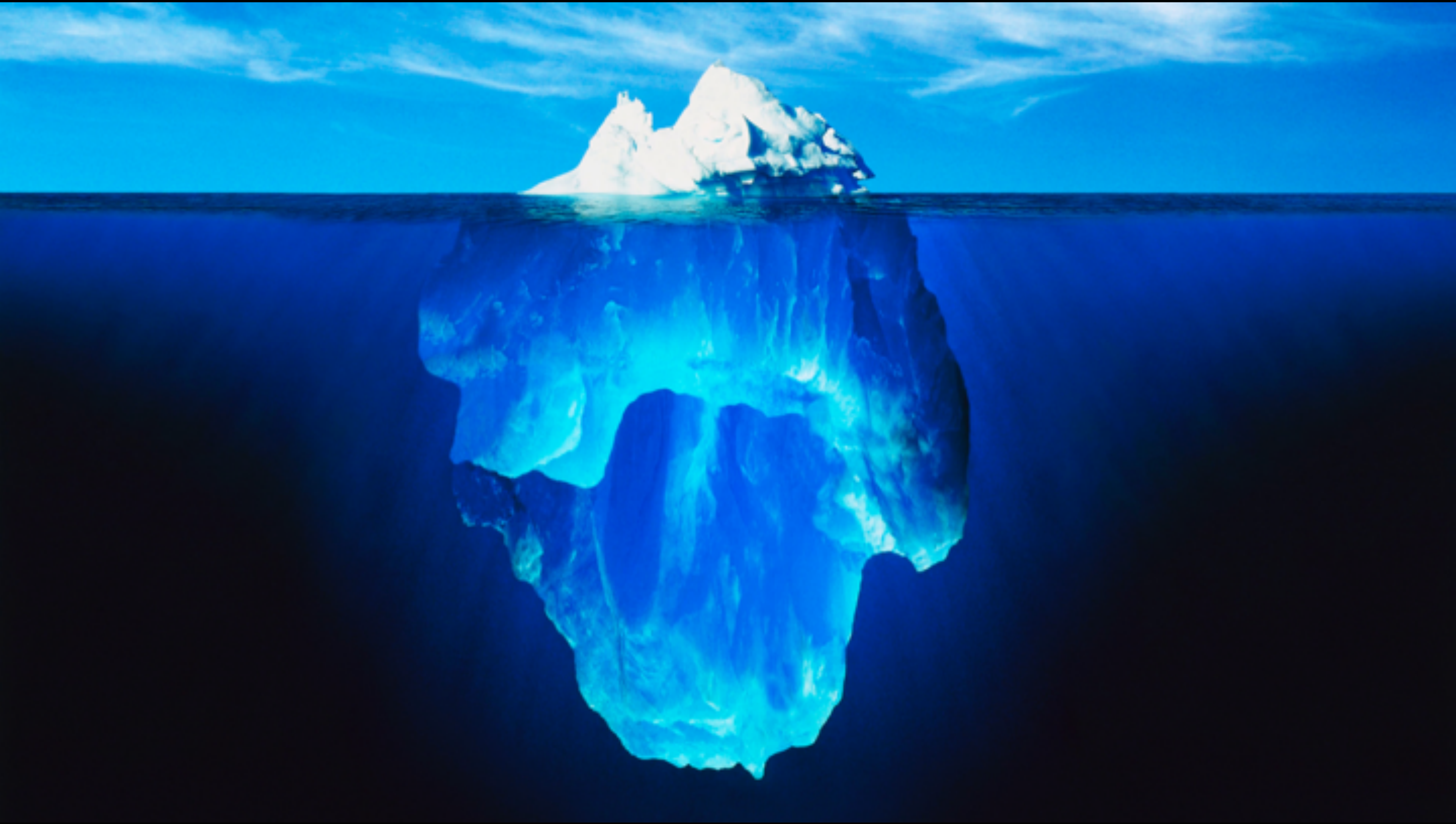
Task-specific sections



Typical usage of yuimaGUI

Task-specific sections

Just the point of the iceberg



This presentation only shows a little portion of both **yuima** and **yuimaGUI** capabilities

Install yuimaGUI !

```
### install dependencies
install.packages("yuima")
install.packages("yuimaGUI")

library(yuima)
library(yuimaGUI)

yuimaGUI() # runs the Shiny GUI
```

Thank You

University of Milan

Contacts

Twitter @iacus

WWW http://www.voices-int.com

Mail stefano.iacus@unimi.it

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI ECONOMIA, MANAGEMENT E METODI QUANTITATIVI