Survey of modelling stochastic differential equations with **R** package *yuima* and implementation of an extended CARMA model

Noah Steidle

Advisor: Niklas Gotthardt

June 1, 2021

Abstract

The **R** package *yuima* enables to simulate and estimate general stochastic processes based on stochastic differential equations. This paper presents the basic operation of *yuima* with a focus on diffusion processes and Lèvy Continuous Autoregressive Moving Average (CARMA) models, which are an important tool for understanding time series and are widely used in financial applications. In addition, a modified CARMA model with an extended drift coefficient is introduced together with a **R** implementation, which simulates trajectories of this model based on given specifications.

Contents

1 Introduction to Stochastic Differential Equations and Numerical Solutions

1.1 Stochastic Differential Equations

The following introduction is loosely based on the lecture *Stochastic Differential Equations* by Prof. Dr. Blömker at the university Augsburg in the winter term 2020/21.

Many processes in nature are described by differential equations in order to define a relationship between physical quantities and their rates of change. In reality these descriptions are perturbed

and vary from measurement to measurement, hence randomness has to be considered in order to find a suitable model and derive useful results.
One possible model is given by stochastic differential equations (SDEs) which generalise the theory of ordinary differential equations (ODEs) by adding a notion of chance. In practice, this randomness is represented by a diffusion function $b(X(t), t)$ and a description of noise ξ which both are added to an ODE

$$\begin{cases} X'(t) = a(X(t), t) + b(X(t), t)\xi(t, \omega), \\ X(0) = 0. \end{cases} \tag{1}$$

The noise ξ in equation 1 has to be defined properly, and after some considerations, one finds $\xi = \partial_t W(t)$, where $W(t)$ is a standard Wiener process.

Definition 1. *A **Standard Wiener process**, often called **Brownian Motion**, is a Gaussian process $W = \{W(t) \mid t \geq 0\}$ with independent increments for which*

1. $W(0) = 0$ *w.p.1,*
2. $\mathbb{E}[W(t)] = 0$,
3. $\mathrm{Var}(W(t) - W(s)) = t - s$,

for all $0 \leq s \leq t$.

In Def. 1 a process is said to have **independent increments** if $W(t) - W(s)$ and $W(u) - W(v)$ are independent whenever $(s, t) \cap (v, u) = \emptyset$ for $s < t,\ v < u$.

Having found a proper description of noise, it is possible to start from equation 1 and get

$$\frac{\partial X}{\partial t} = a(X(t), t) + b(X(t), t)\frac{\partial W(t)}{\partial t},$$

which finally results in the standard notion of a stochastic differential equation

$$dX_t = a(X_t, t)dt + b(X_t, t)dW_t.$$

Exemplary, in the one-dimensional case with $a, b : \mathbb{R} \times \mathbb{R}_+ \to \mathbb{R}$ and $W = (W_t)_{t \geq 0}$ a Standard Wiener process. Another approach to interpret this formula is the integral equation

$$X_t = X_0 + \int_0^t a(X_\tau, \tau)d\tau + \int_0^t b(X_\tau, \tau)dW_\tau.$$

This short introduction is intended to help understanding the following and most implications here should be understood formally. Precise reasoning (e.g. definition of stochastic integrals) can be found for example in [5].

1.2 The Euler-Maruyama scheme

The following explanations are based on [4].
In practical applications stochastic differential equations are often not explicitly solvable, respectively a lot of mathematical considerations are necessary in before. Therefore efficient numerical methods for SDEs have been systematically developed for the last fifty years. To this day, the most efficient and widely applicable approach to solving SDEs has been the simulation of sample paths of time discrete approximations. Given a finite discretization

$$t_0 = \tau_0 < \tau_1 < \cdots < \tau_n < \cdots < \tau_N = T \tag{2}$$

of a time interval $[t_0, T]$, values of sample paths are approximated step by step. Afterwards statistical methods can be used to determine the relation between approximation and exact solution.

We shall consider an Itō process[1] $X = \{X_t \mid t_0 \leq t \leq T\}$ satisfying the stochastic differential equation

$$dX_t = a(t, X_t)dt + b(t, X_t)dW_t$$

on $t_0 \leq t \leq T$ with initial value

$$X_{t_0} = X_0.$$

Given the time discretization 2, an **Euler-Maruyama approximation**, or simply **Euler approximation**, is a continuous time stochastic process $Y = \{Y(t) \mid t_0 \leq t \leq T\}$ satisfying

$$Y_{n+1} = Y_n + a(\tau_n, Y_n)(\tau_{n+1} - \tau_n) + b(\tau_n, Y_n)(W_{\tau_{n+1}} - W_{\tau_n}) \tag{3}$$

for $n = 0, 1, 2, \ldots, N-1$ with initial value

$$Y_0 = X_0.$$

This iterative scheme can also be written

$$Y_{n+1} = Y_n + a(\tau_n, Y_n)\Delta_n + b(\tau_n, Y_n)\Delta W_n,$$

where $\Delta_n = \tau_{n+1} - \tau_n$ and $\Delta W_n = W_{\tau_{n+1}} - W_{\tau_n}$. Sometimes the initial condition is left unstated.

When the diffusion coefficient b is zero, the given scheme 3 reduces to the deterministic Euler scheme. The randomness is anchored in the increments of a Wiener process $W = \{W_t \mid t \geq 0\}$ and can be derived from a sequence of independent Gaussians with the help of a pseudo-random number generator.

Besides the Euler-Maruyama scheme, there exist various schemes to simulate trajectories of a given SDE which all hold different advantages and disadvantages. A series of methods, to which the Euler-Maruyama scheme belongs, are known as *Strong Taylor Approximations*, because there are derived from stochastic Taylor expansions. The different schemes differ in their order of strong convergence. For example the *Milstein scheme*, which also belongs to the Taylor schemes, brings an order of 1.0, whereas the Euler-Maruyama scheme attains an order of strong convergence of 0.5. Another approach is the *Runge-Kutta method*, which grants the advantage of not having to know derivatives of the coefficient functions in the SDE.
Nonetheless, in the following both algorithms presented in *yuima* and new coding examples are based on the Euler-Maruyama scheme.

[1] An Itō process is defined to be an adapted stochastic process that can be expressed in an integral equation.

2 The R Package *yuima*

yuima[2] is an open source **R** package[6], managed and issued by the YUIMA project, which offers various tools to simulate and infer stochastic differential equations. Hereby the stochastic differential equations can be driven by different models, described in the following.

2.1 General Models in *yuima*

First there are Diffusion Models, given by

$$dX_t = \begin{cases} a(t, X_t, \theta)dt + b(t, X_t, \theta)dW_t, \\ X_0 = x_0, \end{cases}$$

where W_t is a standard Brownian motion. These models were introduced in 1.1.
Another model is based on SDEs driven by fractional Gaussian noise, with H the Hurst parameter[2],

$$dX_t = \begin{cases} a(t, X_t, \theta)dt + b(t, X_t, \theta)dW_t^H, \\ X_0 = x_0. \end{cases}$$

At last, the family of diffusion processes with jumps and Lèvy processes can be modelled with *yuima*. These stochastic processes are solutions to

$$dX_t = \begin{cases} a(t, X_t, \theta)dt + b(t, X_t, \theta)dW_t + \int_{|z|>1} c(X_{t-}, z)\mu(dt, dz) \\ \qquad\qquad + \int_{0<|z|\leq 1} c(X_{t-}, z)\{\mu(dt, dz) - \nu(dz)dt\}, \\ X_0 = x_0. \end{cases}$$

In the upcoming the focus will be on Diffusion models only. The basic idea of model creation in *yuima* is presented and exemplary shown for a Diffusion model.

2.1.1 The `yuima.model` class

The most important class in the *yuima* package is the `yuima.model` class. In particular, the constructor function `setModel` enables to define a model with certain mathematical inputs. By calling `setModel` a `yuima.model` object can be created given the following parameters.

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

Hereby `drift` and `diffusion` denote the drift coefficient vector and the diffusion coefficient matrix of the given model. They both are specified by **R** expressions. The `hurst` parameter indicates the Hurst value of the gaussian noise; in particular, the standard value `hurst = 0.5` stands for a Wiener process. Other parameters let one specify jump coefficients and the regarding Lèvy measure, as well as use a prescribed notation of variable names for the SDE. By `xinit` the initial value of the stochastic differential equation can be submitted.

In order to simulate a trajectory of the model `mod`, the *yuima* package offers the method `simulate`. The `simulate` methods accepts inputs of the type `yuima-class`, `yuima.model-class` or `yuima.carma-class` and fills the two slots `data` and `sampling` of the respective `yuima` object, based on already specified properties.

[2]Fractional Brownian motion is a generalization of Brownian motion. In opposition to classical Brownian motion, here the increments need not to be independent and the covariance function is defined by $\mathbb{E}\, B_H(t)B_H(s) = \frac{1}{2}(|t|^{2H} + |s|^{2H} - |t-s|^{2H})$.

Exemplary, regard the following one-dimensional diffusion process

$$dX_t = -5X_t dt + \frac{2}{X_t^2 - 2} dW_t.$$

Following the notation from the previous section, this is a SDE with drift coefficient $a(x) = -5x$ and diffusion coefficient $\frac{2}{x^2-2}$. Translating those coefficients in **R** expressions, the stochastic differential equation can be described in *yuima* as follows.

```
library("yuima")

mod <- setModel(drift = "-5 * x", diffusion = "2/(x^2 - 2)")
```

The variable `mod` now stores a `yuima.model` object. The method `str` gives an overview of all slots of `mod`.

```
str(mod)
## Formal class 'yuima.model' [package "yuima"] with 16 slots
##   ..@ drift          :  expression((-5 * x))
##   ..@ diffusion      :List of 1
##   .. ..$ :  expression((2/(x^2 - 2)))
##   ..@ hurst          : num 0.5
##   ..@ jump.coeff     : list()
##   ..@ measure        : list()
##   ..@ measure.type   : chr(0)
##   ..@ parameter      :Formal class 'model.parameter' [package "yuima"] with 7 slots
##   .. .. ..@ all      : chr(0)
##   .. .. ..@ common   : chr(0)
##   .. .. ..@ diffusion: chr(0)
##   .. .. ..@ drift    : chr(0)
##   .. .. ..@ jump     : chr(0)
##   .. .. ..@ measure  : chr(0)
##   .. .. ..@ xinit    : chr(0)
##   ..@ state.variable : chr "x"
##   ..@ jump.variable  : chr(0)
##   ..@ time.variable  : chr "t"
##   ..@ noise.number   : num 1
##   ..@ equation.number: int 1
##   ..@ dimension      : int [1:6] 0 0 0 0 0 0
##   ..@ solve.variable : chr "x"
##   ..@ xinit          :  expression((0))
##   ..@ J.flag         : logi FALSE
```

In order to simulate a trajectory of the model `mod`, the *yuima* package offers the method `simulate`.

```
X <- simulate(mod)
```

By calling `plot` the trajectory can be visualized. Figure 1 shows the result.

```
plot(X)
```

2.2 Lèvy CARMA models in *yuima*

One further possibility in the *yuima* package is the description of Lèvy CARMA processes [3].

2.2.1 The CARMA model

Initially, the Continuous Autoregressive Moving Average (CARMA) model was derived from discrete-time ARMA processes as a continuous counter part. Until today CARMA processes play an important role, especially in finance.

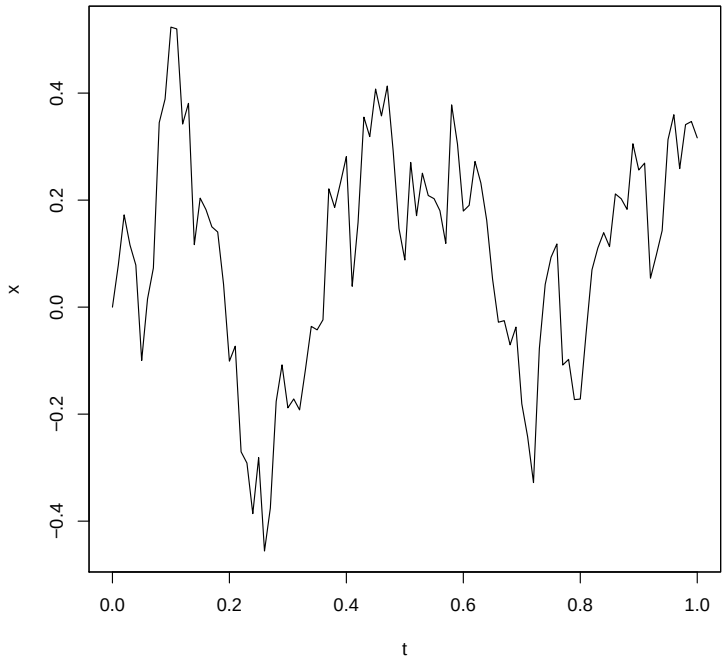


Figure 1: A trajectory of the simulated Diffusion process.

Definition 2. *For p and q non-negative integers with $p > q \geq 0$ the* CARMA(p, q) *process is defined as*

$$a(D)Y_t = b(D)DL_t, \tag{4}$$

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

$$a(z) = z^p + a_1 z^{p-1} + \cdots + a_p,$$
$$b(z) = b_0 + b_1 z + \cdots + b_{p-1} z^{p-1}.$$

Here $a_1, \ldots, a_p$ and $b_0, \ldots, b_{p-1}$ are coefficients such that $b_q \neq 0$ and $b_j = 0 \ \forall j > q$.

Here, L_t can be be a general Lèvy process, but in the upcoming sections L_t will be restricted to a Standard Wiener process due to simplification.
Since the derivatives $D^j L_t$ of higher order do not exist in the usual sense, the describing definition of a CARMA(p, q) process can be specified in a state space representation [1]

$$Y_t = b^T X_t,$$

where X_t of dimension p satisfies the following system of stochastic differential equations

$$dX_t = AX_t dt + e dL_t.$$

Here A is a $p \times p$ matrix given as

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_p & -a_{p-1} & -a_{p-2} & \cdots & -a_1 \end{pmatrix} \in \mathbb{R}^{p \times p},$$

while e and b are $p \times 1$ vectors

$$e = (0, \dots, 0, 1)^T \in \mathbb{R}^p,$$
$$b = (b_0, \cdots, b_{p-1})^T \in \mathbb{R}^p.$$

2.2.2 The `yuima.carma` class

In the *yuima* package exist several methods to describe and work with CARMA(p, q) processes. Most important hereby is the constructor function `setCarma`, which returns an object of type `yuima.carma`. Since `yuima.carma-class` inherits from `yuima.model-class`, methods like `simulate` work very similiar as to described in the previous chapter.
An object of the class `yuima.carma` stores all information about the general linear state space model of the CARMA(p, q) process. The following equations describe the general model and its variables.

$$Y_t = c_0 + \sigma(b^T X_t)$$
$$dX_t = AX_t dt + e(\gamma_0 + \gamma^T X_t) dZ_t,$$

where $c_0 \in \mathbb{R}$ and $\sigma \in (0, \infty)$. $\gamma_0 \in \mathbb{R}$ and $\gamma = [\gamma_1, \dots, \gamma_p]$ are called linear parameters, which play an important role in defining the COGARCH(p, q) model.

Analogously to `setModel`, the `setCarma` function let one specifiy a Carma model. A `yuima.carma` object can be specified in the following way.

```
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 and q define the variable of the CARMA(p, q) model. `loc.par` and `scale.par` describe the label of the location coefficient and the label of the scale coefficient, where standard value `NULL` refers to $c_0 = 0$ and $\sigma = 1$. Autoregressive coefficients a and moving average coefficients b can be set with `ar.par` and `ma.par`. Finally, `Carma.var` and `Latent.var`, default to `"v"` and `"x"`, label the observed process and the unobserved process, while remaining variables enable small changes in the setup, which shall be not focused on here.

As an example regard a CARMA(p, q) model driven by a Wiener process W_t, which explicitly can be written in the state space form like follows.

$$Y_t = b_0 X_{0,t} + b_1 X_{1,t}$$
$$dX_{0,t} = X_{1,t} dt$$
$$dX_{1,t} = X_{2,t} dt$$
$$dX_{2,t} = [-a_3 X_{0,t} - a_2 X_{1,t} - a_1 X_{2,t}] dt + dW_t$$

Based on this specification `setCarma` can be called using the code below.

```
library("yuima")

modC <- setCarma(p=3, q=1, Carma.var="Y", Latent.var="X")
```

Once again `str` displays the content of the slots.

```
str(modC)
## Formal class 'yuima.carma' [package "yuima"] with 17 slots
##   ..@ info          :Formal class 'carma.info' [package "yuima"] with 10 slots
##   .. .. ..@ p         : num 3
##   .. .. ..@ q         : num 1
##   .. .. ..@ loc.par   : chr(0)
##   .. .. ..@ scale.par : chr(0)
```

```
##   .. .. ..@ ar.par    : chr "a"
##   .. .. ..@ ma.par    : chr "b"
##   .. .. ..@ lin.par   : chr(0)
##   .. .. ..@ Carma.var : chr "Y"
##   .. .. ..@ Latent.var: chr "X"
##   .. .. ..@ XinExpr   : logi FALSE
##   ..@ drift          :  expression((b0 * X1 + b1 * X2), (X1), (X2)) ...
##   ..@ diffusion      :List of 4
##   .. ..$ :  expression((0))
##   .. ..$ :  expression((0))
##   .. ..$ :  expression((0))
##   .. ..$ :  expression((1))
##   ..@ hurst          : num 0.5
##   ..@ jump.coeff     : list()
##   ..@ measure        : list()
##   ..@ measure.type   : chr(0)
##   ..@ parameter      :Formal class 'model.parameter' [package "yuima"] with 7 slots
##   .. .. ..@ all      : chr [1:5] "b0" "b1" "a3" "a2" ...
##   .. .. ..@ common   : chr(0)
##   .. .. ..@ diffusion: chr(0)
##   .. .. ..@ drift    : chr [1:5] "b0" "b1" "a3" "a2" ...
##   .. .. ..@ jump     : chr(0)
##   .. .. ..@ measure  : chr(0)
##   .. .. ..@ xinit    : chr(0)
##   ..@ state.variable : chr [1:4] "Y" "X0" "X1" "X2"
##   ..@ jump.variable  : chr(0)
##   ..@ time.variable  : chr "t"
##   ..@ noise.number   : int 1
##   ..@ equation.number: int 4
##   ..@ dimension      : int [1:6] 5 0 0 5 0 0
##   ..@ solve.variable : chr [1:4] "Y" "X0" "X1" "X2"
##   ..@ xinit          :  expression((0), (0), (0)) ...
##   ..@ J.flag         : logi FALSE
```

In a next step the *yuima* package expects a specification of parameters a and b. In order to have a second order solution, the autoregressive coefficients have to be chosen such that the eigenvalues of the matrix A are real and negative [3]. Therefore we choose $a_1 = 6$, $a_2 = 4$ and $a_3 = 6$ for a and $b_0 = 1$ and $b_1 = 0.5$ for b. The model parameters are now fixed by the following statement.

```
par.modC<-list(a1=6,a2=4,a3=6,b0=1,b1=0.5)
```

By specifiying a sampling scheme one gets more control about the simulation.

```
samp<-setSampling(Terminal=400, n=16000)
```

Finally, everything comes together in the `simulate` method. After calling this function `sim.modC` contains the simulated trajectory.

```
sim.modC<-simulate(modC, true.parameter=par.modC, sampling=samp)
```

These trajectories can be visualized via `plot`. A result can be seen in figure 2.

```
plot(sim.modC)
```

3 An extended CARMA Model

3.1 Definition of the new CARMA Model

Consider the following model which is extended by an additive factor in contrast to the definition in previous sections.

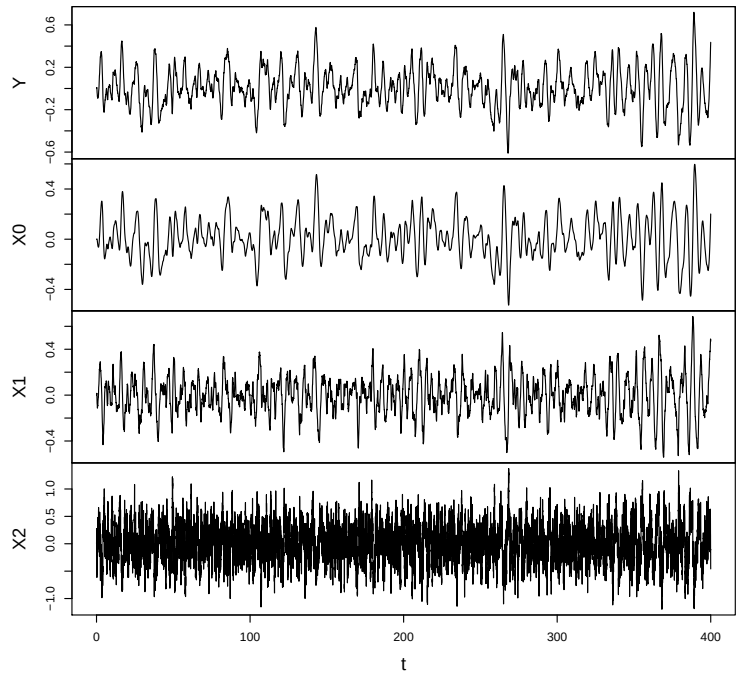


Figure 2: Trajectories of the simulated CARMA process in extityuima.

$$
\begin{aligned}
Y(t) &= b^T X(t) \\
dX(t) &= [AX(t) + CZ(t)]\, dt + e_p dW(t)
\end{aligned}
$$

where

$$
b = \begin{pmatrix} b_0 \\ \vdots \\ b_{p-1} \end{pmatrix} \in \mathbb{R}^p,\ Z(t) = \begin{pmatrix} Z_0(t) \\ \vdots \\ Z_{r-1}(t) \end{pmatrix} \in \mathbb{R}^r,\ e_p = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \in \mathbb{R}^p,
$$

$$
A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_p & -a_{p-1} & -a_{p-2} & \cdots & -a_1 \end{pmatrix} \in \mathbb{R}^{p\times p},\ C = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ -c_r & \cdots & -c_1 \end{pmatrix} \in \mathbb{R}^{p\times r},
$$

with $b_0, \ldots, b_{p-1}$ real-valued coefficients with $b_q = 1$ and $b_j = 0$ for all $j > q$, and $p > q \geq 0$. If $p = 1$ the matrix A is to be understood as $A = (-a_1)$. $(W(t))_{t\geq 0}$ is a standard Wiener process.

3.2 Implementation of the new CARMA model

The **R** function `euler_maruyama_for_new_sde` allows to simulate the modified CARMA model, which was introduced in section 3.1, with an Euler-Maruyama scheme. The entire implementation can be found in the appendix A.
Possible parameters of this function are described here.

```
euler_maruyama_for_new_sde(p, q, as, bs, cs, Z, X0,
                           S=0, T=1, N=1000)
```

`p` and `q` describe the non-negative integers p and q, `as`, `bs` and `cs` are the inputs for the vectors $a = \begin{pmatrix} -a_p & \dots & -a_1 \end{pmatrix}$, $b = \begin{pmatrix} b_0 & \dots & b_{p-1} \end{pmatrix}^T$ and $c = \begin{pmatrix} -c_r & \dots & -c_1 \end{pmatrix}$, respectively. Furthermore, `Z` stands for the vector-valued function $Z(t)$ in $\mathbb{R}^r$, while `X0` is the initial value. `S` and `T` define the underlying time interval, which defaults to $[0, 1]$, with `N` one can define the discretization of the interval $[S, T]$. This is important for the Euler-Maruyama algorithm and is set to 1000 by default.

Assume that we want to build the new model given the following specifications. For $p = 2$ and $q = 1$ consider the coefficient vectors

$$\begin{aligned} a &= \begin{pmatrix} -a_1 & -a_2 \end{pmatrix} = \begin{pmatrix} -3 & -4 \end{pmatrix}, \\ b_0 &= \begin{pmatrix} 2 \end{pmatrix}, \\ c &= \begin{pmatrix} -c_1 & -c_2 \end{pmatrix} = \begin{pmatrix} -2 & -1 \end{pmatrix}, \end{aligned}$$

the initial value $X0 = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$, and vector-valued function Z defined as

$$Z(t) = \begin{pmatrix} t^2 & t^2 \end{pmatrix}^T.$$

Also, we keep the standard interval $[0, 1]$, but increase the number of discretization intervals to $N = 4000$.

In **R** this model can be created and saved in a variable `sim` via the following.

```
a <- c(-3,-4)
b <- c(2)
c <- c(-2,-1)

X0 <- c(0,0)

Z <- function(t) {return(c(t^2,t^2))}

sim <- euler_maruyama_for_new_sde(p=2, q=1, as=a, bs=b, cs=c, Z=Z, X0=X0, N=4000)
```

With the help of `str` one can examine the structure of the result `sim`.

```
str(sim)
## List of 3
##  $ time               : num [1:4001] 0 0.00025 0.0005 0.00075 0.001 0.00125 0.0015 0.00175 0.002 0.00225 ...
##  $ observed_process   : num [1:4001] 0 -0.003821 -0.011537 0.002966 -0.000317 ...
##  $ unobserved_processes: num [1:2, 1:4001] 0.00 0.00 0.00 -3.82e-03 -9.55e-07 ...
```

`sim` references a list which contains three named elements `time`, `observed_process` and `unobserved_process`. Here, `time` represents the time discretization of the interval, `observed_process` is a vector containing the values of Y at each time step, and `unobserved_process` is a matrix consisting of the calculated trajectories $\{X_1, X_2, X_3\}$.

The function `plot_new_sde` enables to visualize these trajectories and the observed process Y. The implementation of `plot_new_sde`, which is based on *ggplot2*[7], can be found in the appendix A as well. Consider the output in 3.

```
plot_new_sde(sim)
```

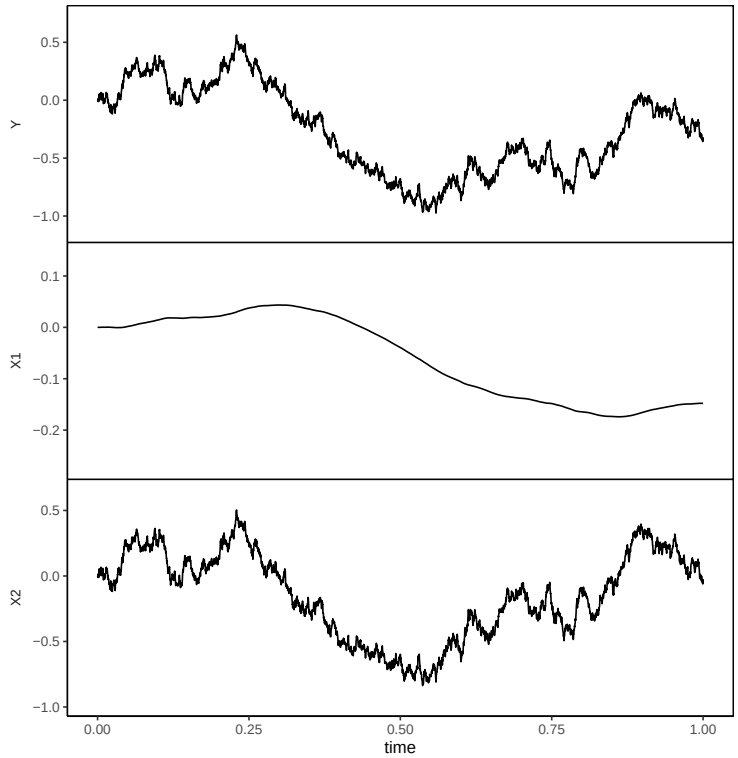


Figure 3: Trajectories of the extended CARMA model.

References

[1] Peter Brockwell. "Lévy-Driven Carma Processes". In: *Annals of the Institute of Statistical Mathematics* 53 (Feb. 2001), pp. 113–124. DOI: 10.1023/A:1017972605872.

[2] Alexandre Brouste et al. "The YUIMA Project: A Computational Framework for Simulation and Inference of Stochastic Differential Equations". In: *Journal of Statistical Software* 57.4 (2014), pp. 1–51. DOI: 10.18637/jss.v057.i04. URL: http://www.jstatsoft.org/v57/i04/.

[3] Stefano M. Iacus and Lorenzo Mercuri. *Implementation of Lévy CARMA model in Yuima package*. 2014. arXiv: 1409.3027 [stat.CO].

[4] Peter E. Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*. Springer Berlin Heidelberg, 1992. DOI: 10.1007/978-3-662-12616-5. URL: https://doi.org/10.1007/978-3-662-12616-5.

[5] Bernt Øksendal. *Stochastic Differential Equations*. Springer Berlin Heidelberg, 2003. DOI: 10.1007/978-3-642-14394-6. URL: https://doi.org/10.1007/978-3-642-14394-6.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2018. URL: https://www.R-project.org/.

[7] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

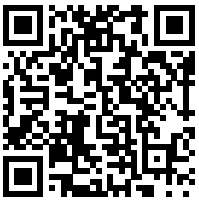
A Implementation

```
euler_maruyama_for_new_sde <- function(p, q, as, bs, cs, Z, X0, S = 0, T = 1, N = 1000) {

    # Assert that input values fit the given model and its restrictions
    if (q >= p)
        stop("'p' must be bigger than 'q'.")
    if (length(as) != p)
        stop("'as' must have dimension 'p'.")
    if (length(bs) != q)
        stop("'bs' must have dimension 'q'.")
    if (length(cs) != length(Z(0)))
        stop("'cs' and 'Z' must have the same dimension.")
    if (length(X0) != p)
        stop("'X0' must have dimension 'p'.")

    # Define length of time discretization subinterval
    tau <- (T - S)/N

    # Define time vector
    times <- seq(from = S, to = T, length.out = N + 1)

    # Build vector b for state space representation
    b <- c(bs, 1, rep(0, p - (q + 1)))

    # Define empty matrix X to store the results
    X <- matrix(data = NA, nrow = p, ncol = N + 1)

    # Set initial value
    X[, 1] <- X0

    # Approximate the SDE - Euler-Maruyama scheme
    for (i in 2:(N + 1)) {

        for (j in 1:p) {

            if (j != p) {
                X[j, i] <- X[j, i - 1] + (tau * X[j + 1, i - 1])
            } else {
                X[j, i] <- X[j, i - 1] + (tau * (sum(rev(as) * X[, i - 1]) + sum(rev(cs) *
                  Z(times[i - 1])))) + sqrt(tau) * rnorm(1)
            }

        }

    }

    # Apply vector b for state space representation
    Y = colSums(b * X)

    # Save results to list
    result_list <- list(times, Y, X)

    # Name columns of the list
    names(result_list) <- c("time", "observed_process", "unobserved_processes")

    # Return (invisible) list
    return(invisible(result_list))

}
```

Implementation of `euler_maruyama_for_new_sde` in **R**

```
plot_new_sde <- function(sim) {

    # Get p and N from data input
    p <- dim(sim$unobserved_processes)[1]
    N <- dim(sim$unobserved_processes)[2]

    # Convert input to dataframe
    df <- data.frame(value = c(t(sim$unobserved_processes)), label = rep(paste("X",
        1:p, sep = ""), each = N), time = rep(sim$time, p))

    # Add observed process Y to dataframe
    df <- rbind(df, data.frame(value = sim$observed_process, label = rep("Y", N),
        time = sim$time))

    # Save labels as factor
    df$label <- factor(df$label, levels = c("Y", paste("X", 1:p, sep = "")))

    # Call ggplot
    ggplot(df, aes(x = time, y = value)) +
    geom_line() + labs(y = "") + scale_y_continuous(expand = c(0.1, 0.1)) +
    facet_grid(label ~ ., switch = "y", scales = "free_y") +
    theme_classic() + theme(strip.background = element_blank(), strip.placement = "outside",
        panel.spacing = unit(0, "lines"), panel.border = element_rect(color = "black",
            fill = NA))

}
```

Implementation of `plot_new_sde` in **R**



A github repository containing the implementation can be found here.