Euler–Maruyama method

In Itô calculus, the **Euler–Maruyama method** (also called the **Euler method**) is a method for the approximate numerical solution of a stochastic differential equation (SDE). It is a simple generalization of the Euler method for ordinary differential equations to stochastic differential equations. It is named after Leonhard Euler and Gisiro Maruyama. Unfortunately, the same generalization cannot be done for any arbitrary deterministic method [1].

Consider the stochastic differential equation (see Itô calculus)

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

with initial condition $X_0 = x_0$, where W_t stands for the Wiener process, and suppose that we wish to solve this SDE on some interval of time $[0, T]$. Then the **Euler–Maruyama approximation** to the true solution X is the Markov chain Y defined as follows:

- partition the interval $[0, T]$ into N equal subintervals of width $\Delta t > 0$:

$$0 = \tau_0 < \tau_1 < \cdots < \tau_N = T \text{ and } \Delta t = T/N;$$

- set $Y_0 = x_0$;
- recursively define Y_n for $1 \le n \le N$ by

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

where

$$\Delta W_n = W_{\tau_{n+1}} - W_{\tau_n}.$$

The random variables ΔW_n are independent and identically distributed normal random variables with expected value zero and variance Δt.

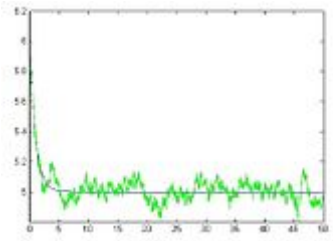
Contents

Example

Numerical simulation

An area that has benefited significantly from SDE is biology or more precisely mathematical biology. Here the number of publications on the use of stochastic model grew, as most of the models are nonlinear, demanding numerical schemes.

The graphic depicts a stochastic differential equation being solved using the Euler Scheme. The deterministic counterpart is shown as well.



Gene expression modelled as stochastic process

Computer implementation

The following Python code implements the Euler–Maruyama method and uses it to solve the Ornstein–Uhlenbeck process defined by

$$dY_t = \theta \cdot (\mu - Y_t)\,\mathrm{d}t + \sigma\,\mathrm{d}W_t$$

$$Y_0 = Y_{\text{init}}.$$

The random numbers for $\mathrm{d}W_t$ are generated using the NumPy mathematics package.

```python
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

num_sims = 5  # Display five runs

t_init = 3
t_end  = 7
N      = 1000  # Compute 1000 grid points
dt     = float(t_end - t_init) / N
y_init = 0

c_theta = 0.7
c_mu    = 1.5
c_sigma = 0.06

def mu(y, t):
    """Implement the Ornstein-Uhlenbeck mu."""  # = \theta (\mu-Y_t)
    return c_theta * (c_mu - y)

def sigma(y, t):
    """Implement the Ornstein-Uhlenbeck sigma."""  # = \sigma
    return c_sigma

def dW(delta_t):
    """Sample a random number at each call."""
    return np.random.normal(loc=0.0, scale=np.sqrt(delta_t))

ts    = np.arange(t_init, t_end, dt)
ys    = np.zeros(N)

ys[0] = y_init

for _ in range(num_sims):
    for i in range(1, ts.size):
        t = (i-1) * dt
        y = ys[i-1]
        ys[i] = y + mu(y, t) * dt + sigma(y, t) * dW(dt)
    plt.plot(ts, ys)

plt.show()
```

The following is simply the translation of the above code into the MATLAB (R2019b) programming language:

```
%% Initialization and Utility
close all;
 3 clear all;

numSims = 5;            % display five runs
tBounds = [3 7];        % The bounds of t
N       = 1000;         % Compute 1000 grid points
dt      = (tBounds(2) - tBounds(1)) / N ;
y_init = 1;             % Initial y condition


pd = makedist('Normal',0,sqrt(dt)); % Initialize the probability distribution for our
                         % random variable with mean 0 and
                         % stdev of sqrt(dt)

c = [0.7, 1.5, 0.06];   % the initial Theta, Mu, and Sigma, respectively

ts     = linspace(tBounds(1), tBounds(2), N); % From t0-->t1 with N points
ys     = zeros(1,N);     % 1xN Matrix of zeros

ys(1) = y_init;
%% Computing the Process
for j = 1:numSims
    for i = 2:numel(ts)
        t = (i-1) .* dt;
        y = ys(i-1);
        mu      = c(1) .* (c(2) - y);
        sigma   = c(3);
        dW      = random(pd);

        ys(i) = y + mu .* dt + sigma .* dW;
    end
    figure(69)
    hold on;
    plot(ts, ys, 'o')
end
```

See also

- Milstein method
- Runge–Kutta method (SDE)

References

1. Kloeden, P.E. & Platen, E. (1992). *Numerical Solution of Stochastic Differential Equations*. Springer, Berlin. ISBN 3-540-54062-8.