Comparison of the Euler-Maruyama Method and Milstein Method for Stochastic Differential Equations of Robotic Systems

Guangda Xu
(Student Number: A0268548L)
College of Design and Engineering
National University of Singapore

Abstract

This paper undertakes a computational performance comparison of the Euler-Maruyama Method and Milstein's Higher Order Method, for solving a certain kind of Stochastic Differential Equation of a robotic system. The comparison mainly focuses on accuracies, time consumptions, and memory requirements for different scales of problems (i.e. different stepsizes). In this paper, the implementation of the two methods is based on the MATLAB codes provided in Desmond J. Higham's paper. The primary goal of this paper is to provide an empirical assessment of the relative computational efficiency of the two methods under varying problem conditions.

Key Words: Euler-Maruyama Method, Milstein's Higher Order Method, Stochastic Differential Equation, MATLAB

1 Introduction

Stochastic Differential Equations (SDEs) are mathematical models used to describe a system that changes over time and also experiences some random fluctuations. In robotics, SDEs are often used to model the behavior of robots operating in stochastic environments with disturbances such as noises. By incorporating stochastic factors into the dynamic and control algorithms, the robot's ability to interact with the changing environment can be improved [1].

The Euler-Maruyama (EM) Method and Milstein's Higher Order Method are both numerical methods commonly used for solving SDEs. Analytically, for the same Itô stochastic differential equation

$$\mathrm{d}X_t = a(X_t)\mathrm{d}t + b(X_t)\mathrm{d}W_t \tag{1}$$

The Euler–Maruyama approximation to the true solution X can be recursively defined as

$$X_{n+1} = X_n + a(X_n, \tau_n)\Delta t + b(X_n, \tau_n)\Delta W_n \tag{2}$$

In contrast, the Milstein approximation involves higher-order expansion. That is,

$$X_{n+1} = X_n + a(X_n)\Delta t + b(X_n)\Delta W_n + \frac{1}{2} b(X_n) b'(X_n)((\Delta W_n)^2 - \Delta t) \tag{3}$$

For both of the two approximations, $0 \leq n \leq N - 1$.

It is generally believed that the main difference between these two methods lies in the convergence order. The Euler-Maruyama is a first-order method and often simpler, however, the Milstein method is second-order and more accurate for small stepsize problems [2]. In this paper, several numerical experiments are conducted using MATLAB to compare their computational performance in application.

2 Description of the Problem Set

In this paper, we consider the stochastic dynamics of a certain robot, which can be described by

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t = \alpha X(t)dt + \beta X(t)dW_t,\ X(0) = X_0,\ 0 \leq t \leq T. \tag{4}$$

Here we set $\alpha = 3$, $\beta = 0.5$, and $X_0 = 0.5$, for the convenience of doing numerical simulation.

It is known [3] that the exact solution to this SDE is

$$X(t) = X(0) \exp\left(\left(\alpha - \frac{1}{2}\beta^2\right)t + \beta W(t)\right) = 0.5 \exp\big(2.875t + 0.5W(t)\big) \tag{5}$$

This solution will be used for subsequent accuracy analysis.

3 MATLAB Codes and Explanation of Modifications

According to the requirement of performance comparison, we have made some modifications to the MATLAB codes given by Higham [4].

3.1 The Euler-Maruyama Method

In this part, in order to compare the accuracy under different stepsizes, an additional layer of `for` loop is introduced into the main part of the code. Furthermore, to optimize computational efficiency, the value of `R` was converted to a vectorized expression (i.e., a `1-by-4` array), similar to the approach used in the Milstein method. A printer mechanism for errors under different stepsizes is also embedded.

In addition, a function for recording memory usage `monitor_memory_whos()`, which will be explained in detail in section 4.2, has been introduced.

The modified code is shown in the M-file `EMNew.m` in Listing 1.

```
% EMNEW Euler-Maruyama method on linear SDE
%
% SDE is dX = alpha*X dt + beta*X dW, X(0) = Xzero,
% where alpha = 2, beta = 1 and Xzero = 1.
%
% Discretized Brownian path over [0,1] has dt = 2^(-8).
% Euler-Maruyama uses timestep R*dt.
randn('state',100)
alpha = 3; beta = 0.5; Xzero = 0.5; % problem parameters
T = 1; N = 2^8; dt = 1/N;
tic;
dW = sqrt(dt)*randn(1,N); % Brownian increments
W = cumsum(dW); % discretized Brownian path
Xtrue = Xzero*exp((alpha-0.5*beta^2)*([dt:dt:T])+beta*W);
R = [2,4,8,64]; % L EM steps of size Dt = R*dt
for num=1:4
    Dt = R(num)*dt; L = N/R(num);
    Xem = zeros(1,L); % preallocate for efficiency
    Xtemp = Xzero;
    for j = 1:L
        Winc = sum(dW(R(num)*(j-1)+1:R(num)*j));
        Xtemp = Xtemp + Dt*alpha*Xtemp + beta*Xtemp*Winc;
        Xem(j) = Xtemp;
    end
    emerr(num) = abs(Xem(end)-Xtrue(end));
end
solvertime=toc
memory_usage = monitor_memory_whos()
emerr_1=emerr(1)
emerr_2=emerr(2)
emerr_3=emerr(3)
emerr_4=emerr(4)

% =========Memory Record Function=========

function [ memory_in_use ] = monitor_memory_whos( )

% This function uses the 'whos' command and evaluates inside the base
% workspace and sums up the bytes.  The output is displayed in MB.
```

```
mem_elements = evalin('base','whos');
if size(mem_elements,1) > 0

    for z = 1:size(mem_elements,1)
        memory_array(z) = mem_elements(z).bytes;
    end

    memory_in_use = sum(memory_array);
    memory_in_use = memory_in_use/1048576;
else
    memory_in_use = 0;
end

end
```

Listing 1 *M-file* `EMNew.m`

3.2 Milstein's Higher Order Method

The code provided in Higham's paper regarding this method generally meets the requirements of this paper. As such, only a few slight adjustments were made, including altering the SDE expression and incorporating mechanisms for outputting error values, in addition to introducing the function for memory usage recording, as shown in the M-file `MilNew.m` in Listing 2.

To conserve space, some repetitive comments from section 3.1 have been omitted.

```
% MILNEW Milstein's Higher Order Method - vectorized
%
% Solves dX = alpha*X dt + beta*X dW, X(0) = Xzero,
% where alpha = 3, beta = 0.5 and Xzero = 0.5.
%
% Discretized Brownian path over [0,1] has dt = 2^(-8).
% Milstein uses timesteps 64*dt, 8*dt, 4*dt, 2*dt (also dt for reference).
%
% Code is vectorized: all paths computed simultaneously.
rand('state',100)
alpha = 3; beta = 0.5; Xzero = 0.5; % problem parameters
T = 1; N = 2^(8); dt = T/N; %
M = 500; % number of paths sampled
R = [1; 2; 4; 8; 64]; % Milstein stepsizes are R*dt

tic;
dW = sqrt(dt)*randn(M,N); % Brownian increments
Xmil = zeros(M,5); % preallocate array
for p = 1:5
    Dt = R(p)*dt; L = N/R(p); % L timesteps of size Dt = R dt
    Xtemp = Xzero*ones(M,1);
    for j = 1:L
        Winc = sum(dW(:,R(p)*(j-1)+1:R(p)*j),2);
        Xtemp = Xtemp + Dt*alpha*Xtemp + beta*Xtemp.*Winc ...
        + 0.5*beta^2*Xtemp.*(Winc.^2 - Dt);
    end
    Xmil(:,p) = Xtemp; % store Milstein solution at t =1
end


Xref = Xmil(:,1); % Reference solution
Xerr = abs(Xmil(:,2:5) - repmat(Xref,1,4)); % Error in each path
err = mean(Xerr); % Mean pathwise errors
Dtvals = dt*R(2:5); % Milstein timesteps used

solvertime=toc
memory_usage = monitor_memory_whos()
err_R2=err(1)
err_R4=err(2)
err_R8=err(3)
err_R64=err(4)


% =========Memory Record Function=========

function [ memory_in_use ] = monitor_memory_whos( )

mem_elements = evalin('base','whos');
if size(mem_elements,1) > 0

    for z = 1:size(mem_elements,1)
        memory_array(z) = mem_elements(z).bytes;
    end

    memory_in_use = sum(memory_array);
    memory_in_use = memory_in_use/1048576;
else
    memory_in_use = 0;
end
end
```

Listing 2 *M-file* `MilNew.m`

4 Computational Findings

In this section, we conduct a series of comparative evaluation of the computational accuracy and efficiency of the Euler-Maruyama method (hereinafter referred to as EM method) and Milstein method for the selected problem set from the following perspectives.

It is worth noting that in both Listing 1 and 2, we uniformly set `dt=2^(-8)` as the sampling interval for discretized Brownian motion [5]. Then the timesteps used for simulation is of size `Dt = R*dt`, where `R = [2,4,8,64]` is chosen as the test set.

4.1 Accuracy

The accuracy of both methods is evaluated by the value of error, that is, the difference between the program solution and the exact solution. In listing 1 (for EM method), the exact solution (5) is embedded, and the error, denoted as `emerr`, is a `1-by-4` array that stores the errors of solutions under four different timesteps.

In listing 2 (for Milstein method), the error is denoted as `err` (mean pathwise error). The comparison mechanism is similar to Listing 1, while the exact solution is selected as the Milstein solution with $\Delta t = \delta t$ (i.e., when `R=1`), for simplicity.

Throughout the experiment, each program was executed twice. To ensure consistency across other variables, one set of stepsizes was set to be the same in both runs, meaning that the experimental findings encompassed 7 distinct sets of stepsizes.

The error results corresponding to the two different methods are shown in Table 1 and the comparison is shown in Figure 1.

Table 1 Comparison of values of errors

R	2	4	8	16	32	64	128
`EMNew.m`	0.2590	0.7358	0.7302	1.3959	2.3344	3.2034	4.3188
`MilNew.m`	0.1676	0.4870	1.0700	2.0452	3.4724	5.1707	6.7227

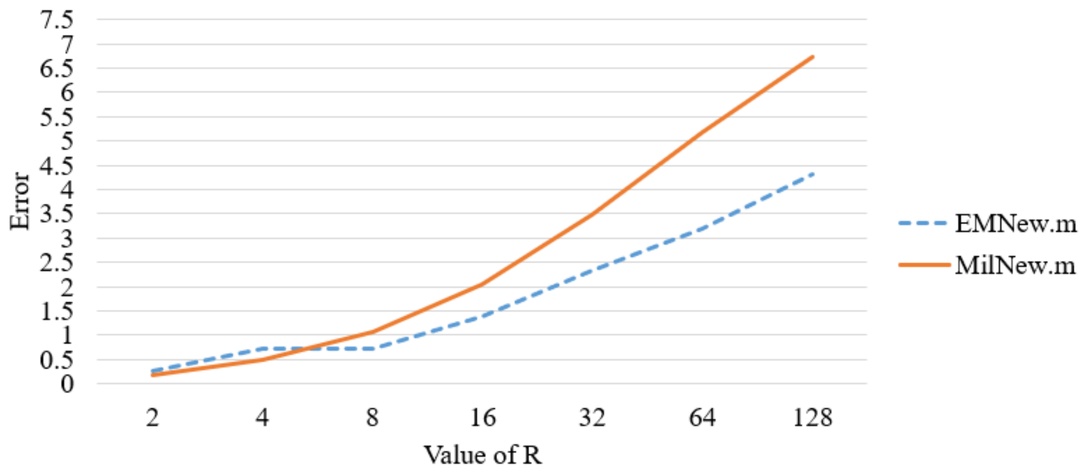

Figure 1 Values of errors under different stepsizes

It can be seen that the error of Milstein method decreases significantly faster with stepsizes than the EM method. For sufficiently small stepsizes, the result of Milstein method tend to be more accurate. In brief, the Milstein method is better suited for computations that demand a high degree of precision, where this approach has demonstrated superior performance.

4.2 Memory Usage

We developed a function `monitor_memory_whos` in our program to track memory usage during the solving process. This function operates by evaluating memory usage within the base workspace and aggregating the bytes occupied, finally display the result in MB. It should be noted that while MATLAB offers a built-in function called `memory`, what its output reflects is the total memory occupied by the MATLAB client [6], which is of limited relevance to our analysis.

Here, both of the two programs contains four stepsizes, as indicated at the beginning of section 4. The memory occupied by the two programs under different sampling intervals for discretized Brownian motion (i.e., `dt`) during operation is shown in Table 2. Each outcome presented herein is the mean derived from three repetitions of the experiment.

Table 2 Comparison of memory usage (MB)

`N=1/dt`	`2^8`	`2^11`	`2^14`
`EMNew.m`	0.0062	0.0473	0.3830
`MilNew.m`	1.0225	7.8585	62.5460
Multiple of difference	$\frac{1.0225}{0.0062}=165$	$\frac{7.8585}{0.0473}=166$	$\frac{62.5460}{0.3830}=163$

It can be seen that as the sampling interval `dt` decreases (`N` increases), the memory occupied by both methods shows an upward trend. In addition, the Milstein method consumes substantially more memory compared to the EM method, which aligns with the description presented in Section 1 elucidating the distinctions between the two methods, that is, the Milstein method is second-order and relatively more complex.

4.3 Time Consumptions

We measured the time consumptions required by the EM and Milstein method to solve the problem set. Similar to section 4.2, it needs to be emphasized that the obtained time consumption includes calculations based on four different stepsizes. The result is shown in Table 3.

Table 3 Comparison of time consumption (s)

`N=1/dt`	`2^8`	`2^11`	`2^14`
`EMNew.m`	0.0043	0.0079	0.0136
`MilNew.m`	0.0285	0.0546	0.3284
Multiple of difference	$\frac{0.0285}{0.0043}=6.6$	$\frac{0.0546}{0.0079}=6.9$	$\frac{0.3284}{0.0136}=24.1$

Given that the time consumption of code running is largely dependent on both the computer configuration and the system operation, to maintain consistency, the experiment was conducted on uniform equipment and under identical running environment. The result was also averaged through multiple runs.

It is evident that for both methods the time consumption increases in tandem with the increase in `N`. Through this comparative analysis, our findings also indicate that both methods have demonstrated high efficiency, but comparatively, the time consumption of the Milstein method is significantly longer. Even though the magnitude of the difference is smaller than that of memory usage in section 4.2, it is still a factor that must be considered in practical applications.

5 Conclusions

In this paper, we conducted a comparative analysis of the computational performance of two numerical methods, the Euler-Maruyama method and Milstein's Higher Order method, for stochastic differential equations. We introduced the stochastic dynamics of a certain robotic system in the form of SDE and modified the MATLAB codes provided in Higham's paper for experimentation.

The analysis of this paper has revealed that the performance of the EM method and Milstein method is timestep-dependent and exhibits some differences. Specifically, the following conclusions have been reached.

(1) **In terms of accuracy.** Both two methods exhibit a reduction in errors as the stepsize decreases, with the Milstein method showing a relatively faster variation. Notably, for instances requiring high precision (small stepsizes), the Milstein method yields superior results.

(2) **In terms of memory usage and time consumption.** The two methods both show a negative correlation between sampling intervals for Brownian motion and memory usage or time consumption. Nevertheless, throughout the program execution, the system resources occupied by Milstein method significantly surpass those of the EM method. This difference becomes more noticeable as `dt` decreases, which highlights the importance of considering memory and time requirements when selecting a method for solving SDEs.

In conclusion, both EM and Milstein are efficient numerical methods for solving stochastic differential equations. However, for high-accuracy required issues, the Milstein method, as a second-order one, is proven to be a better choice. By contrast, the Euler-Maruyama method is characterized by its simplicity, high speed, and less occupation of system memory, making it a favorable alternative.

References

[1] Raffin, A., Kober, J., & Stulp, F. (2022, January). Smooth exploration for robotic reinforcement learning. In *Conference on Robot Learning* (pp. 1634-1644). PMLR.
[2] Tanaka, H., & Yamada, T. (2014). Strong convergence for Euler–Maruyama and Milstein schemes with asymptotic method. *International Journal of Theoretical and Applied Finance*, *17*(02), 1450014.
[3] Mao, X. (2007). *Stochastic differential equations and applications*. Elsevier.
[4] Higham, D. J. (2001). An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM review*, *43*(3), 525-546.
[5] Boudreault, M., & Renaud, J. F. (Eds.). (2019). *Brownian motion* (pp. 325–364). John Wiley & Sons, Inc.
[6] Near, T. (2021). An Analysis into the Performance and Memory Usage of MATLAB Strings. *arXiv preprint arXiv:2109.12567*.