Computational Methods in Finance
Computational project 2

Matej Rojec, Patrycja Ozgowicz, Erik Hamberg

June 27, 2023

Contents

List of Figures

1 Task 1

Write Matlab routines for generating pseudo-random numbers from a standard normal distribution $\mathcal{N}(0, 1)$ using the acceptance-rejection method and the linear congruential generator for obtaining realizations of a uniform distribution over [0, 1].

Firstly, we have written the linear congruential generator for obtaining realizations of a uniform distribution over [0, 1] as a Matlab function. We have decided to use common values for $M = 2^{31} - 1$, $a = 16807$ and $b = 0$. The function named uniform_generator takes two arguments: N, which is the number of uniform realizations and $m1$, which is the seed of the generator. It returns a vector U of realizations of a uniform distribution over [0, 1]. The code is presented below:

```
function [U] = uniform_generator(N,m1)
    M = 2^(31)-1;
    a = 16807;
    b = 0;
    m = zeros(1,N);
    m(1) = m1;
    for i=2:N
        m(i) = mod(a*m(i-1)+b,M);
    end
    U = m/M;
end
```

In order to check if our function works properly, we decided to plot a histogram of one million realizations of uniform distribution over [0, 1] with seed $m_1 = 0.2$ with theoretical density function, which is equal to 1 on the interval [0,1] and 0 elsewhere. The results are presented on the figure 1.

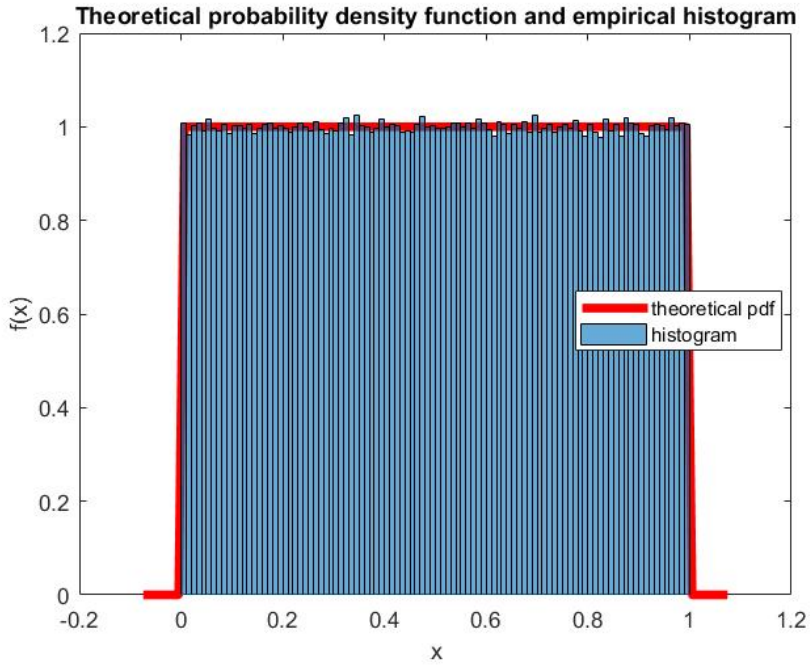

Figure 1: The histogram of uniform realization obtained using uniform_generatior function compared with theoretical density function. We took $N = 1000000$ and $m_1 = 0.2$.

On the plot 1 empirical result (blue histogram) is close to the theoretical density function (red line), so we assumed, that the generator works properly.

Then, we have used the acceptance-rejection method for generating pseudo-random numbers from a standard normal distribution with density function

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}. \tag{1}$$

For this case, we have used another random variable Y, with probability density function $g(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, that we are able to simulate by using the inversion method. We have:

$$G(y) = \int_{-\infty}^{y} g(t)\,dt = \frac{1}{\pi}\int_{-\infty}^{y} \frac{1}{1+t^2}\,dt = \frac{1}{\pi}\arctan(y) + \frac{1}{2}.$$

Therefore,

$$y = G^{-1}(u) = \tan\left(u\pi - \frac{\pi}{2}\right)$$

So, we simulated the random variable Y by taking pseudo-random points defined by $y = \tan\left(u_1\pi - \frac{\pi}{2}\right)$, where u_1 are realizations of a $\mathcal{U}([0,1])$ distribution. Then, we defined $u_2 \sim \mathcal{U}([0,1])$ (independent from u_1) and accepted just

the points that satisfy $u_1 \leq \frac{f(y)}{Mg(y)}$. Here,

$$M = \frac{f(1)}{g(1)} = \sqrt{\frac{2\pi}{e}}$$

is the absolute maximum of $\frac{f(x)}{g(x)}$.

Again, we have decided to write the routine as a Matlab function. It is called normal_generator and takes three arguments: N, which is the number of normal realizations, $m1$ and $m2$, which are the seeds of the uniform generator. It returns a vector X of realizations of a standard normal distribution $\mathcal{N}(0,1)$. Since we know that the probability of acceptance of the method is equal to $\frac{1}{M}$, we have decided to generate $N \cdot M \cdot 2$ realizations from a uniform distribution and then choose the first N accepted results. The code is presented below:

```
M = sqrt(2*pi/exp(1));
new_N = N * ceil(M) * 2;
U1 = uniform_generator(new_N,m1);
Y = tan(U1*pi - pi/2);
U2 = uniform_generator(new_N,m2);
X = Y(U2 <=((1/sqrt(2*pi) * exp(-(Y.^2)/2)).*(M/pi *
    1./(1+Y.^2)) ));
X = X(1:N);
```

In order to check if our function works properly, we decided to plot a histogram of one million realizations of the standard normal distribution $\mathcal{N}(0,1)$ with seeds $m_1 = 0.2$ and $m_2 = 0.5$ with theoretical density function, which is presented in equation 1. The results are presented on the figure 2.

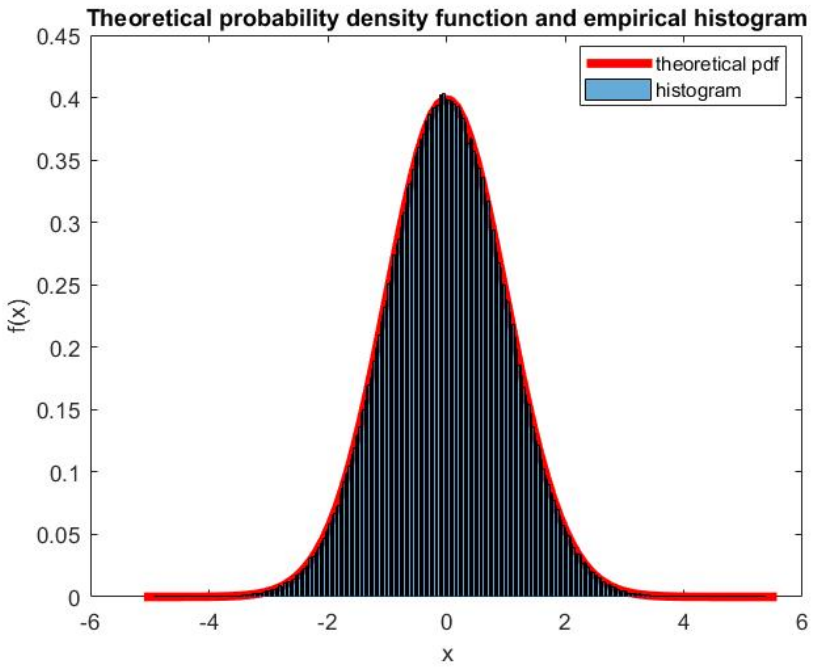

Figure 2: The histogram of standard normal realization obtained using normal_generatior function compared with theoretical density function. We took $N = 1000000$, $m_1 = 0.2$ and $m_2 = 0.5$.

On the plot 2 empirical histogram is really close to the theoretical density function (red line). It indicates that the generator works properly.

2 Task 2

Write Matlab routines of Euler-Maruyama and Milstein methods for solving a general stochastic differential equation (SDE) of type

$$dX(t) = a(t, X(t))dt + b(t, X(t))dB(t), \quad 0 < t \leq T \tag{2}$$

with initial condition $X(0) = x_0 \in \mathbb{R}$ and determine the approximate solution at equally spaced points

$$t_0 = 0, t_1 = h, t_2 = 2h, \ldots, t_N = Nh = T, \text{where } h = \frac{T}{N},$$

for some $N \in \mathbb{N}$.

We have written Matlab function for the Euler-Maruyama method. It takes 7 arguments: a and b, which are the functions mentioned in equation 2, T, which

is the time horizon, N, which indicate the number of time periods, x_0 the initial value and m_1, m_2, which are seeds for generating the normal distribution. Code for the function is presented below:

```
function [lt,x] = Euler_Maruyama_method(a,b,T,N,x0,m1,m2)
    h=T/N;
    lt=0:h:T;
    lx=zeros(size(lt));
    lx(1)=x0;
    Z=normal_generator(N,m1,m2);
    for j=1:N
        lx(j+1) = lx(j) + h*a(lt(j),lx(j)) + b(lt(j),lx(j
            ))*sqrt(h)*Z(j);
    end
    x = lx;
end
```

Then we have written Matlab function for the Milstein method, which is an extension to the previous method. It takes 8 arguments: a and b, which are the functions mentioned in equation 2, diff_b, which is the partial derivative of the function b with respect to X, T, which is the time horizon, N, which indicate the number of time periods, x_0 the initial value and m_1, m_2, which are seeds for generating the normal distribution. Code for the function is presented below:

```
function [lt,x] = Milstein_method(a,b,diff_b,T,N,x0,m1,m2
    )
    h=T/N;
    lt=0:h:T;
    lx=zeros(size(lt));
    lx(1)=x0;
    Z=normal_generator(N,m1,m2);
    for j=1:N
        lx(j+1) = lx(j) + h*a(lt(j),lx(j)) + b(lt(j),lx(j
            ))*sqrt(h)*Z(j) + 0.5*b(lt(j),lx(j)) *diff_b(
            lt(j),lx(j))* ((sqrt(h)*Z(j))^2-h);
    end
    x = lx;
end
```

3 Task 3

Consider the following SDE

$$dS(t) = \mu S(t)dt + \sigma S(t)dB(t), \quad 0 < t \leq T \tag{3}$$

whose solution is given by the geometric Brownian motion,

$$S(t) = S(0)\ \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma B(t)\right) \tag{4}$$

Apply the routines developed in 2. to solve this SDE using Euler-Maruyama and Milstein methods for the following parameters $T = 1$, $\mu = 0.5$ and $\sigma = 0.3$ and present and discuss the following numerical simulations:

a) For a given realization of the Brownian motion, in the same figure plot the exact solution defined by (3) and the numerical solutions obtained by Euler-Maruyama and Milstein methods, taking $h = 0.001$.

b) Consider the following values of $h = 0.005 \times (\frac{1}{2})^i$, $i = 0, 1, 2, 3$ and for each of the values of h, run 500 000 simulations of Euler-Maruyama and Milstein methods and use them to estimate the order of strong convergence and order of weak convergence of both methods.

To create the plot of the three solutions to the equation we use the following script:

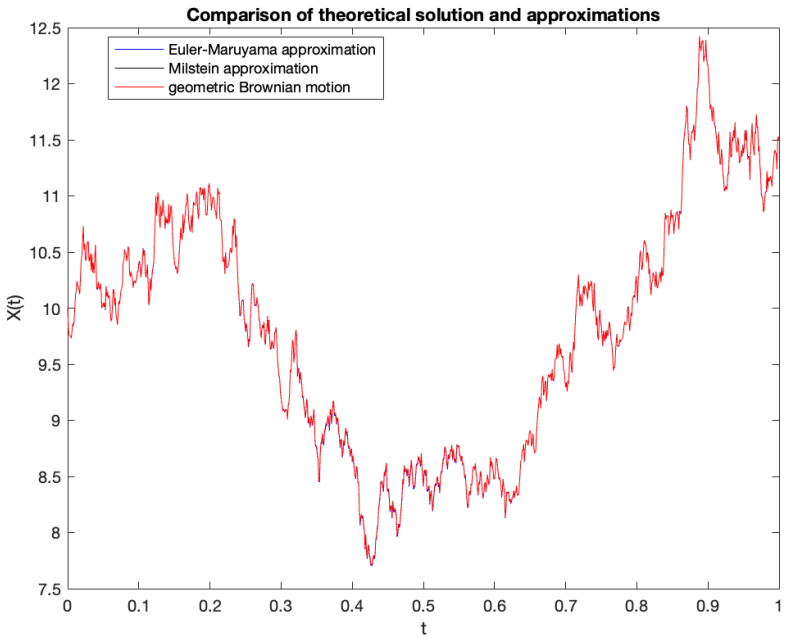

Figure 3: Three different solutions to a realization of Brownian motion

```
T=1;
N=1000;
```

```
x0 = 10;
h=T/N;
mu = 0.5;
sigma = 0.3;
a = @(t,x) mu * x;
b = @(t,x) sigma * x;
diff_b = @(t,x) sigma;
m1 = 0.2;
m2 = 0.5;

[lt,X] = Euler_Maruyama_method(a,b,T,N,x0,m1,m2);
[lt,X2] = Milstein_method(a,b,diff_b,T,N,x0,m1,m2);

Z=normal_generator(N,m1,m2);
S(1)=x0;
B = [0; cumsum(Z')] *sqrt(h);
for j=1:N
  S(j+1) = x0 * exp((mu - (sigma^2)/2) * lt(j+1) + sigma*
     B(j+1));
end

figure(1)
plot(lt,X,'b')
hold on
plot(lt,X2,'k')
plot(lt,S,'r')
legend({'Euler-Maruyama approximation','Milstein
    approximation','geometric Brownian motion'},'Location'
    ,'best')
title('Comparison of theoretical solution and
    approximations')
xlabel('t')
ylabel('X(t)')
```

What we can deduce from the graphs is that the two methods are successful in solving the SDE numerically. There are some minor differences from the exact solution, but the movement is largely the same.

In theory, the solutions should converge with $O(h)$ in weak convergence and $O(h)$ in strong convergence for the Milstein method and $O(h^{0.5})$ in strong convergence and $O(h)$ in weak convergence for the Euler-Maruyama method. We got the same results in our analysis. To find this result, we fit a line to the log errors of the two solutions. By looking at the slope of the fits we can deduce the order of convergence. The results can be seen in table 1 and on figure 4. We see that the theoretical convergence matches that of our experiments.

	Slope	Intercept
Strong Convergence Mil	0.9987	0.9548
Strong Convergence EM	0.5057	-0.1361
Weak Convergence	0.9977	0.7058
Weak Convergence	1.0145	0.8025

Table 1: Convergence analysis of the Millstein and the Euler-Maruyama solutions.

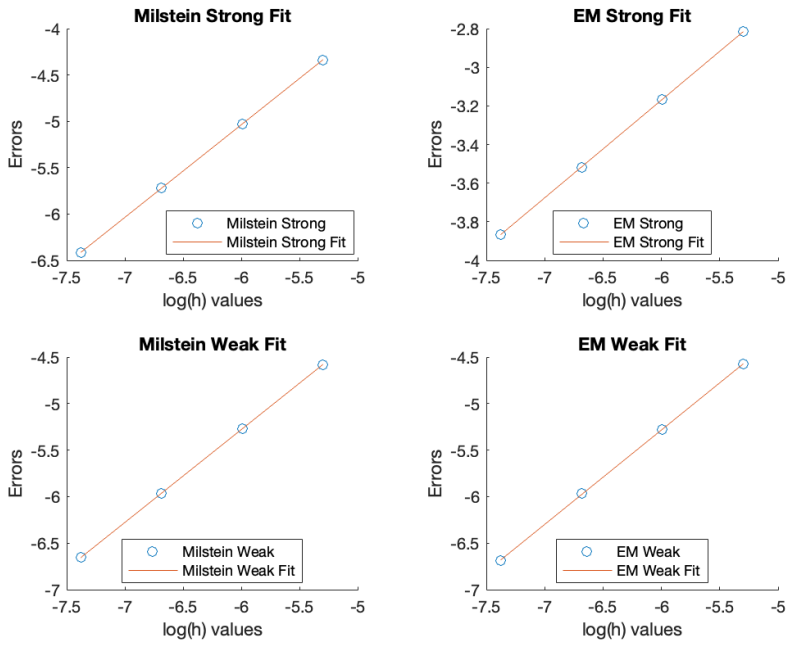

Figure 4: The points and the line fitted to the convergence analysis of our implementations of the Milstein and the Euler-Maruyama method.

The code used to generate this analysis is the following:

```
T=1;
x0 = 10;
mu = 0.5;
sigma = 0.3;
a = @(t,x) mu * x;
b = @(t,x) sigma * x;
diff_b = @(t,x) sigma;
m1 = 0.2521;
m2 = 0.14324;
```

```
h_values = 0.005 * (1 / 2).^ (0:3);
num_simulations = 500000;

orders_strong_EM = zeros(size(h_values));
orders_weak_EM = zeros(size(h_values));
orders_strong_Milstein = zeros(size(h_values));
orders_weak_Milstein = zeros(size(h_values));

errors_Milstein_weak = zeros(size(h_values));
errors_Milstein_strong = zeros(size(h_values));
errors_EM_weak = zeros(size(h_values));
errors_EM_strong = zeros(size(h_values));

for i = 1:length(h_values)
    h = h_values(i);
    N = T / h;
    errors_EM_w = zeros(1, num_simulations);
    errors_Milstein_w = zeros(1, num_simulations);
    errors_EM_s = zeros(1, num_simulations);
    errors_Milstein_s = zeros(1, num_simulations);
    history_S = zeros(1, num_simulations);

    for j = 1:num_simulations
        m1 = m1 - m1/num_simulations;
        m2 = m2 - m2/num_simulations;

        Z=normal_generator(N,m1,m2);
        [lt,X_EM] = Euler_Maruyama_method(a,b,T,N,x0,m1,
            m2);
        [lt,X_M] = Milstein_method(a,b,diff_b,T,N,x0,m1,
            m2);

        Z=normal_generator(N,m1,m2);
        S = zeros(size(X_M));
        S(1)=x0;
        B = [0; cumsum(Z')] *sqrt(h);
        for k=1:N
          S(k+1) = x0 * exp((mu - (sigma^2)/2) * lt(k+1)
              + sigma*B(k+1));
        end

        errors_EM_w(j) = X_EM(end);
        errors_Milstein_w(j) = X_M(end);
        history_S(j) = S(end);
    end
```

```
    disp(i)
    errors_Milstein_weak(i) = abs(mean(errors_Milstein_w
        - history_S));
    errors_EM_weak(i) = abs(mean(errors_EM_w - history_S)
        );
    errors_Milstein_strong(i) = mean(abs(
        errors_Milstein_w - history_S));
    errors_EM_strong(i) = mean(abs(errors_EM_w -
        history_S));
end

errors_log_Milstein_strong = log(errors_Milstein_strong);
errors_log_EM_strong = log(errors_EM_strong);
errors_log_Milstein_weak = log(errors_Milstein_weak);
errors_log_EM_weak = log(errors_EM_weak);

log_h_values = log(h_values);
coefficients_Milstein_strong = polyfit(log_h_values,
    errors_log_Milstein_strong, 1)
coefficients_EM_strong = polyfit(log_h_values,
    errors_log_EM_strong, 1)
coefficients_Milstein_weak = polyfit(log_h_values,
    errors_log_Milstein_weak, 1)
coefficients_EM_weak = polyfit(log_h_values,
    errors_log_EM_weak, 1)

x = linspace(min(log_h_values), max(log_h_values), 100);

fit_Milstein_strong = polyval(
    coefficients_Milstein_strong, x);
fit_EM_strong = polyval(coefficients_EM_strong, x);

fit_Milstein_weak = polyval(coefficients_Milstein_weak, x
    );
fit_EM_weak = polyval(coefficients_EM_weak, x);
```