Report

Brownian Motion Simulation and related models

By QFRM 2 Student: Bondarenko Maksym

Academic Supervisor:

Manolessou Marietta

Contents

INTRODUCTION

Monte Carlo simulation furnishes the decision-maker with a range of possible outcomes and the probabilities they will occur for any choice of action.

The technique was first used by scientists working on the atom bomb; it was named for Monte Carlo, the Monaco resort town renowned for its casinos. Since its introduction in World War II, Monte Carlo simulation has been used to model a variety of physical and conceptual systems.

During a Monte Carlo simulation, values are sampled at random from the input probability distributions. Each set of samples is called an *iteration,* and the resulting outcome from that sample is recorded. Monte Carlo simulation does this hundreds or thousands of times, and the result is a probability distribution of possible outcomes. In this way, Monte Carlo simulation provides a much more comprehensive view of what may happen. It tells you not only what could happen, but how likely it is to happen.

Monte Carlo simulation provides a number of advantages over *deterministic,* or "single-point estimate" analysis:

- *Probabilistic Results.* Results show not only what could happen, but how likely each outcome is.
- *Graphical Results.* Because of the data a Monte Carlo simulation generates, it's easy to create graphs of different outcomes and their chances of occurrence. This is important for communicating findings to other stakeholders.
- *Sensitivity Analysis.* With just a few cases, deterministic analysis makes it difficult to see which variables impact the outcome the most. In Monte Carlo simulation, it's easy to see which inputs had the biggest effect on bottom-line results.

- *Scenario Analysis:* In deterministic models, it's very difficult to model different combinations of values for different inputs to see the effects of truly different scenarios. Using Monte Carlo simulation, analysts can see exactly which inputs had which values together when certain outcomes occurred. This is invaluable for pursuing further analysis.
- *Correlation of Inputs.* In Monte Carlo simulation, it's possible to model interdependent relationships between input variables. It's important for accuracy to represent how, in reality, when some factors goes up, others go up or down accordingly. [1]

SIMULATION OF RANDOM VARIABLES

The method of Polar coordinates in simulation of normal distribution
(Code in [Annex 1])

The *polar plane* consists of a reference axis, or ray, that emanates from a point called the origin. Positions or coordinates are determined according to the distance or radius, from the origin, symbolized R and the angle relative to the reference axis, symbolized by the lowercase Greek theta θ. [3]

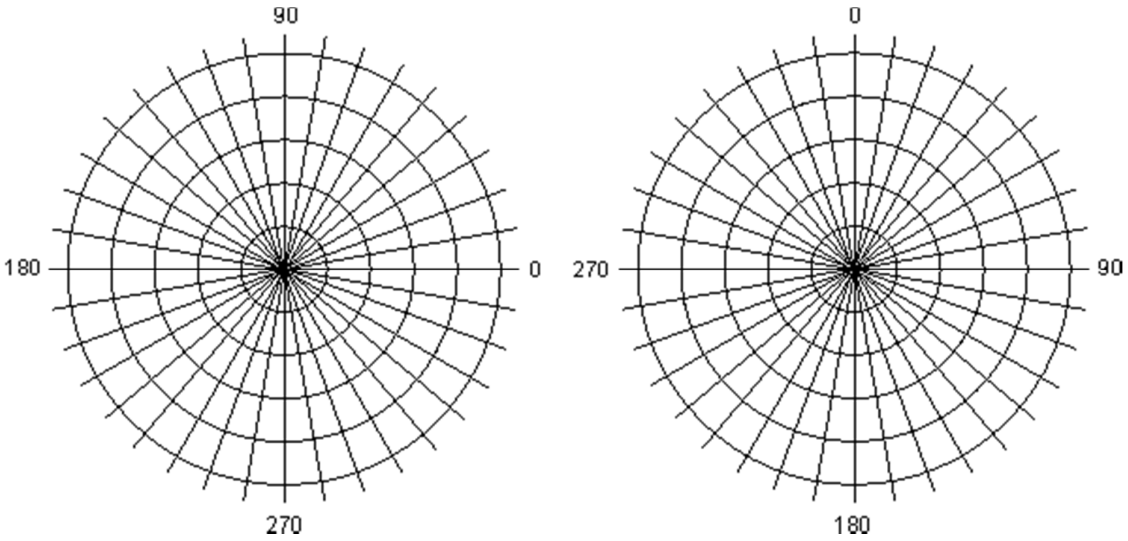


Fig. 1.1 Polar plane [3]

We determine 2 transformations of $U1$ and $U2$ which yield 2 standard gaussian random variables ($X1$, $X2$) by using polar coordinates:

$$\begin{cases} R^2 = X_1^2 + X_2^2 \\ \theta = arctan\dfrac{X_2}{X_1} \end{cases}$$

which are independent if $X1$, $X2$ are independent. [2]

In fact we have obtained two r.v.: a radius and an angle that determine a value of coordinates of a r.v. in polar system of coordinates:

$$\begin{cases} R^2 = -2\ ln\ U1 \text{ (follows exp}\left(\frac{1}{2}\right)) \\ \theta = 2\pi U2\ (follows\ Uniform\ on\ [0, 2\pi]) \end{cases}$$

And:

$$\begin{cases} X_1 = Rcos(\theta) \\ X_2 = Rsin(\theta) \end{cases}$$

So:

$$\begin{cases} X1 = (-2\ ln\ U1)1/2\ cos(2\pi U2) \\ X2 = (-2\ ln\ U1)1/2\ sin(2\pi U2) \end{cases}$$

But to reduce the time of calculations we use these facts:

$2U - 1$ Uniform on $[-1, 1]$ and let

$$\begin{cases} V_1 = 2U_1 - 1 \\ V_2 = 2U_2 - 1 \end{cases}$$

Then we simulate a sequence of random variables U_1, U_2 and V_1, V_2 consequently. We choose only the pairs where $V_1 + V_2 \leq 1$ so that they were in the circle with

R = 1/2 and center at (0,0).

Now

$$\begin{cases} \sin(\theta) = \frac{V_2}{R} = -\frac{V_2}{\sqrt{V_1^2 + V_2^2}} \\ \cos(\theta) = \frac{V_1}{R} = -\frac{V_1}{\sqrt{V_1^2 + V_2^2}} \end{cases}$$

Now we put these expressions into an expression for two independent random variables that will follow $N(0, 1)$:

$$\begin{cases} X = (-2\ln(R^2))^{1/2}\frac{V_1}{R} = \sqrt{\frac{-2\ln(S)}{S}}V_1 \\ Y = (-2\ln(R^2))^{1/2}\frac{V_2}{R} = \sqrt{\frac{-2\ln(S)}{S}}V_2 \end{cases}$$

where $S = R^2 = V_1^2 + V_2^2$

Procedure of the Simulation [2]

Step 1: Generate the random numbers $U1$ and $U2$

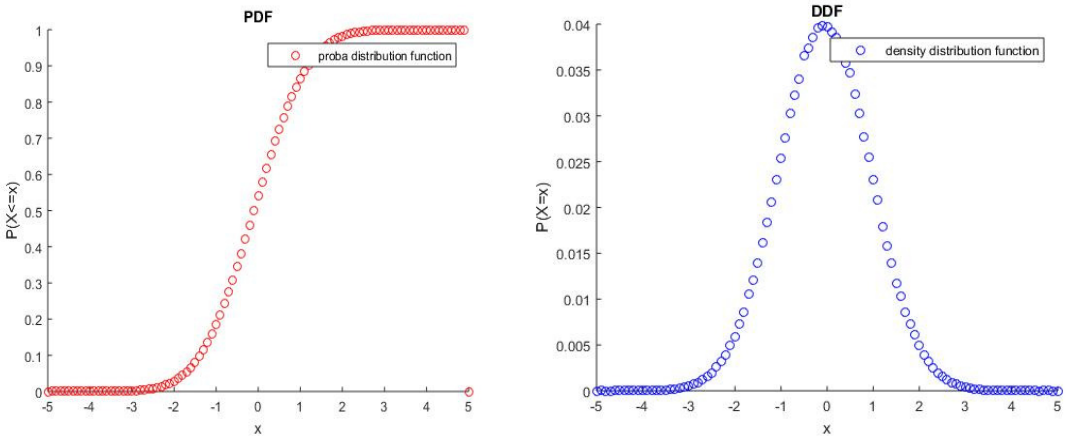
Step 2: Put $V1 = 2U1 - 1$, $V2 = 2U2 - 1$, $S = V_1^2 + V_2^2 = R^2$

Step 3: If $S > 1$ come back to step 1

Step 4: Simulate the 2 independent standard gaussian random variables

Experimental results:

The result of 1 million iterations: E(X) = -0.0011; Var(X) = 1.0012



The method of reject in simulation of normal distribution
(Code in [Annex 2])

The principle [2]

a) We know *how to simulate* a continuous random variable Y with a density distribution function: $g(y)$ and,

b) By using the density $g(y)$, we want to simulate another random variable X with known density distribution function $f(x)$ but with not explicitly given the corresponding probability distribution function FX.

c) We first simulate Y and accept this value with a probability proportional to the ratio $f(y)/g(y)$;

$\Leftrightarrow$ given a previously defined constant C we require $\forall y, f(y), g(y) \leq C$

The steps [2]:

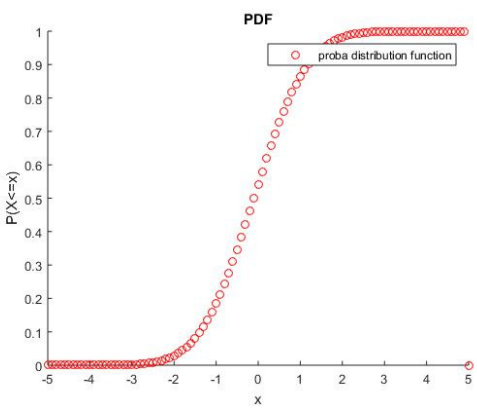
1. We simulate $Y = y$ and calculate $f(y)$, $g(y)$
2. We simulate a random variable *Random* $U = u$
3. If $u \leq f(y)\ Cg(y)$ we put $x = y$

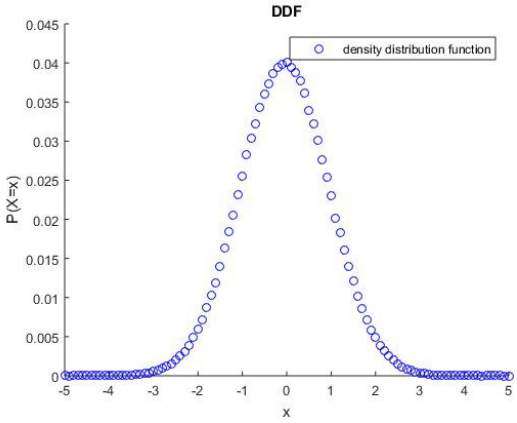
   If not, we go back to step 1.

Experimental results:

The result of 1 million iterations: E(X) = 2.1396e-04 = 0; Var(X) = 1.0000

But in new experiments it gives the same precision as Polar method.


PDF
proba distribution function
P(X<=x)
x


DDF
density distribution function
P(X=x)
x

TCL method in simulation of normal distribution

(Code in [Annex 3])

Theorem 3.1 (T.C.L.) *Let* (Ω, *A, P*), *be a probability space and let us consider a sequence of independent random variables X*1, *X*2, . . . *Xn defined on* (Ω, *A*), *which follow the same probability distribution law and such that:* $\forall i \in \{1, 2, \ldots n\}$

E[*Xi*] = μ, *Var*[*Xi*] = σ2

*exist the sequence Y*1 , *Y*2, . . *Yn*, . . . *where*

$$Yn = \sqrt{n} \times \frac{\bar{X} - \mu}{\sigma} = \frac{1}{\sqrt{n}} \times \sum_{i=1}^{n} \frac{X_i - \mu}{\sigma}$$

converges in law to a standard normal random variable Y : *N*(0, 1) [2]

For this Simulation we apply the "Central Limit Theorem" by using a sample of Continuous Uniform random variables on]0,1[, $\Leftrightarrow$ (*X*1 , . . . , *Xn*) = (*U*1 , . . . , *Un*) Then, taking into account the facts that:

E [U] = 1/2;

Var [U] = 1/12;

The **"Central Limit Theorem"** implies that for sufficiently large *n* the variable

$$Y_n = \sqrt{n} \times \frac{\bar{U} - 0.5}{\frac{1}{\sqrt{12}}} = \frac{1}{\sqrt{n}} \times \sum_{i=1}^{n} ((U_i - 0.5) \times \sqrt{12})$$

follows the standard Gaussian law: *Y* : *N*(0, 1) [2]

So, to simulate an r.v. by this method we:

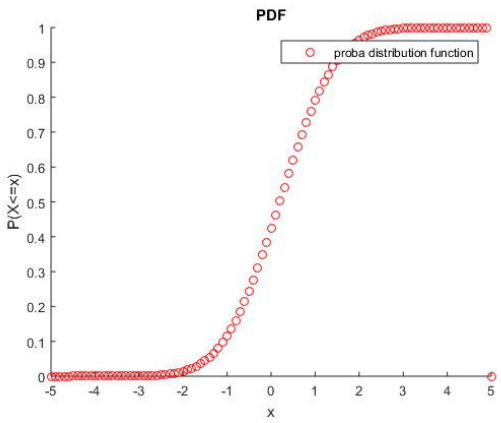
1) Find an arithmetic mean of uniform random variables
2) Center and normalize it obtaining normal centered r.v. by TCL

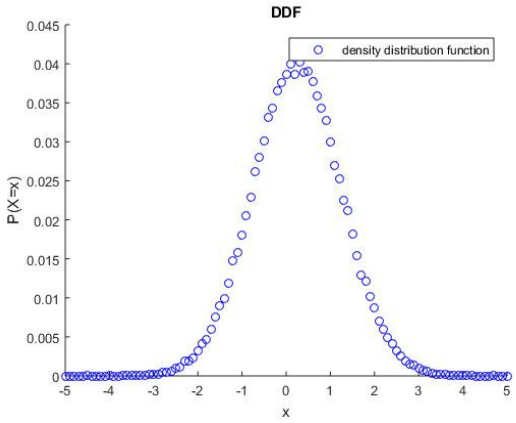
Experimental results:

The result of 100000 iterations: E(X) = 0.2955; Var(X) = 0.9981

We had to do less iterations as this method is much slower than two other.

The reason for this – 1000 of additional operations on every iteration (as method assumes simulating random numbers to obtain their arithmetic mean).





SIMULATION OF THE BROWNIAN MOTION: RANDOM WALK AND GAUSSIANS

(Code in [Annex 4])

1st method: Random walk (multiple trajectories)

Justification:

We know (from definition of Brownian Motion):

$$Y = \frac{W_t - 0}{\sqrt{t}} \quad (1)$$

From TCL using the fact that

$E[S_{[nt]}] = 0$; $Var[S_{[nt]}] = 1/[nt]$:

$$Y_n = \lim_{n \to \infty} \frac{S_{[nt]}}{\sqrt{n}\sqrt{t}} = Y: N(0,1)$$

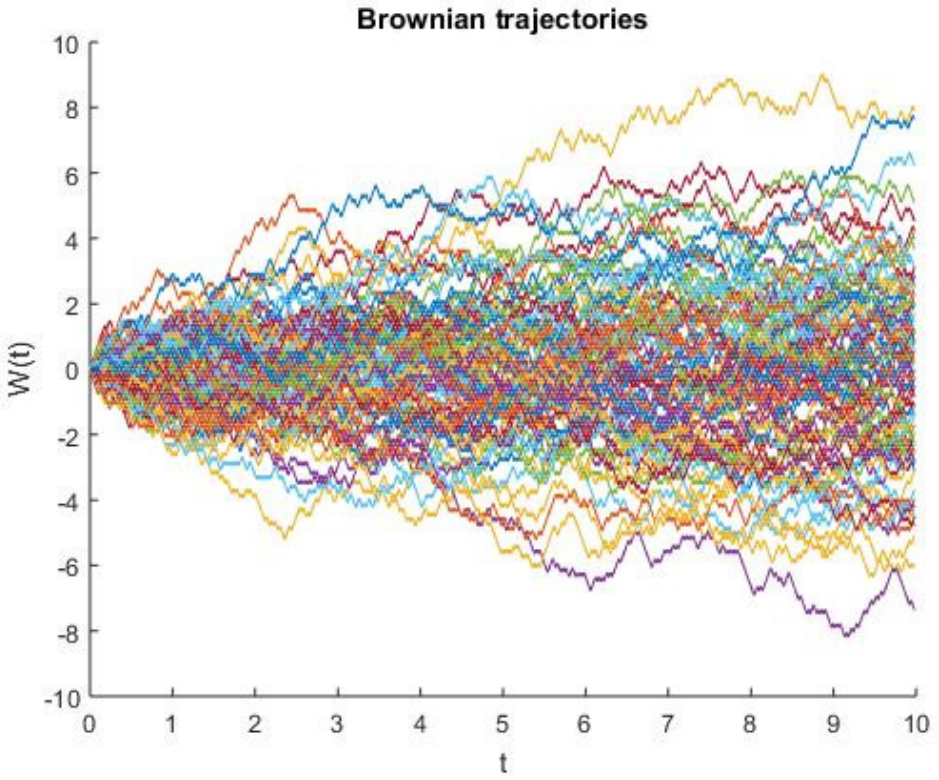
Then putting Y into (1) we obtain an approximation of Brownian Motion for sufficiently large n:

$$W_t = \frac{S_{[nt]}}{\sqrt{n}}$$

Experimental results:

Initial data: T = 10; Discretization of t = 500; $F_0 = \{W0 = W(1) = 0\}$; Number of trajectories = 100;

Results for t = T = 10: $E(W_{10}|F_0) = 0.5171$ (theoretical = $T = W_0 = 0$) ; $Var(W_{10}|F_0) = 10.4277$ (theoretical = T =10)


Brownian trajectories
W(t)
t
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
1
2
3
4
5
6
7
8
9
10

2nd method: Gaussians

Justification:

Let us define

$$\bar{S}_n = \sum_{i=0}^{n-1} \frac{g_i}{n}$$

From TCL using the fact that

$$\begin{cases} E[\overline{S_n}] = 0 \\ Var[\overline{S_n}] = \frac{1}{n} \end{cases}$$

$$\Rightarrow \lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{g_i}{\sqrt{i}} = Z\text{: } N(0,1)$$

Now

$$\frac{W_{n\Delta t}}{\sqrt{n\Delta t}} = Z$$

So

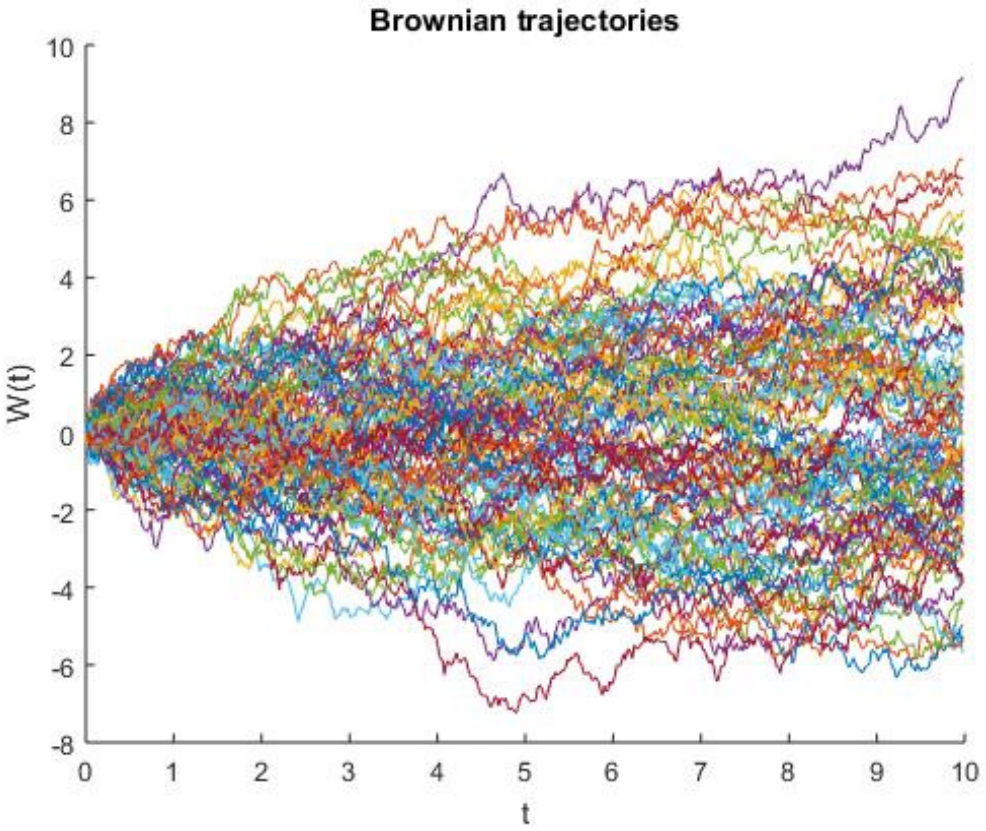
$$W_{n\Delta t} = Z\sqrt{n\Delta t} \quad or \quad W_t = S_n\sqrt{\Delta t}$$

Experimental results:

2.1 Polar coordinates:

Initial data: T = 10; Discretization of t = 500; F_0 = {W0 = W(1) = 0}; Number of trajectories = 100;

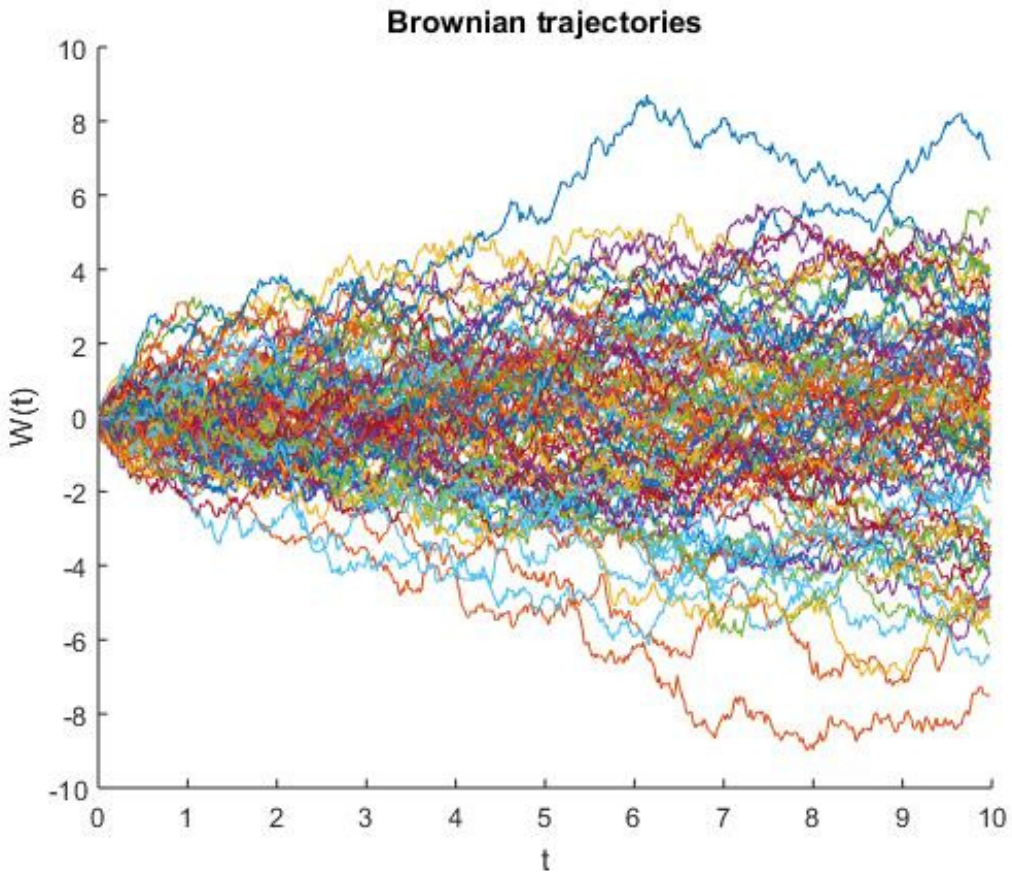
Results for t = T = 10: $E(W_{10}|F_0)$ = -0.0777 (theoretical = T = W_0 = 0) ; $Var(W_{10}|F_0)$ = 9.3131 (theoretical = T =10)



2.2 Reject method

Initial data: T = 10; Discretization of t = 500; F_0 = {W0 = W(1) = 0}; Number of trajectories = 100;

Results for t = T = 10: $E(W_{10}|F_0)$ = -0.1380 (theoretical = T = W_0 = 0) ; $Var(W_{10}|F_0)$ = 8.7037 (theoretical = T =10)

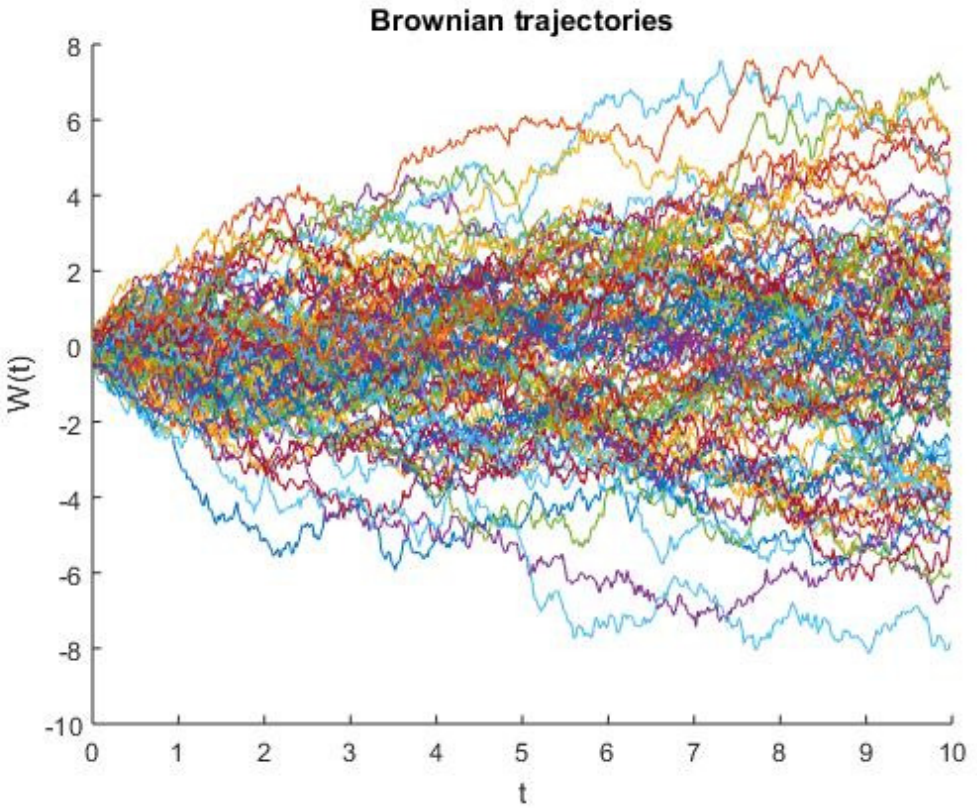


2.3 TCL method

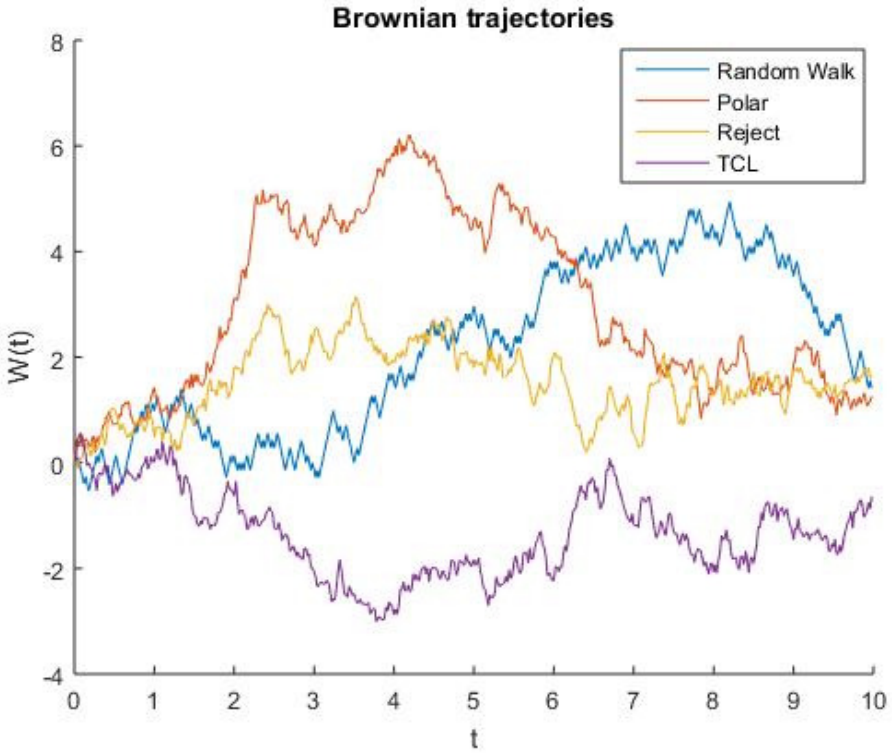
Initial data: T = 10; Discretization of t = 500; F_0 = {W0 = W(1) = 0}; Number of trajectories = 100;

Results for t = T = 10: $E(W_{10}|F_0)$ = -0.1123 (theoretical = T = W_0 = 0) ; $Var(W_{10}|F_0)$ = 9.1784 (theoretical = T =10)

+ Slower execution



Comparison of 4 trajectories:


Brownian trajectories
Random Walk
Polar
Reject
TCL
W(t)
t
8
6
4
2
0
-2
-4
0
1
2
3
4
5
6
7
8
9
10

RELATED MODELS SIMULATION

Geometric Brownian Motion (Asset price simulation):

(Code in [Annex 5])

A geometric Brownian motion (GBM) (also known as exponential Brownian motion) is a continuous-time stochastic process in which the logarithm of the randomly varying quantity follows a Brownian motion (also called a Wiener process) with drift. It is an important example of stochastic processes satisfying a stochastic differential equation (SDE); in particular, it is used in mathematical finance to model stock prices in the Black–Scholes model.

SDE:

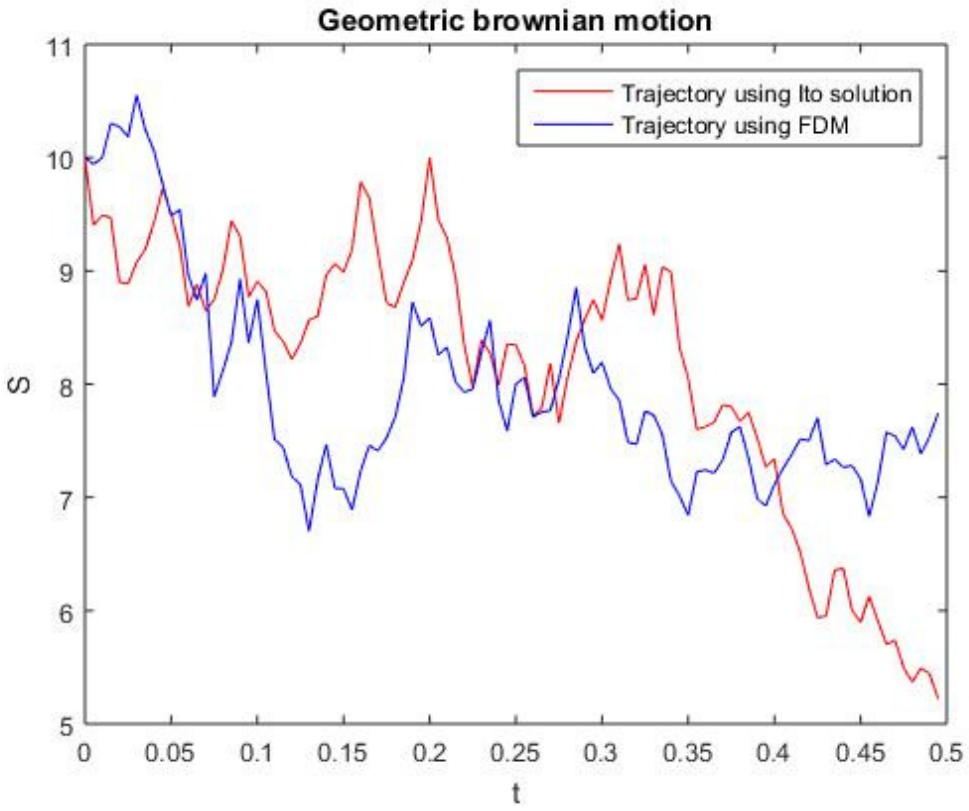
$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

Analytical solution (Ito):

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right)$$

Underlying functions: BMsimulator, Normal_polar

Initial data: S0 = 10; mu = 0.1; sigma = 0.5; T = 0.5; delta_t = 0.005;


Geometric brownian motion
Trajectory using Ito solution
Trajectory using FDM
S
t
11
10
9
8
7
6
5
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5

Ornstein-Ulenbeck (Interest rate simulation):

(Code in [Annex 6])

In mathematics, the Ornstein–Uhlenbeck process (named after Leonard Ornstein and George Eugene Uhlenbeck), is a stochastic process that, roughly speaking, describes the velocity of a massive Brownian particle under the influence of friction. The process is stationary Gauss–Markov process (which means that it is both a Gaussian and Markovian process), and is the only nontrivial process that satisfies these three conditions, up to allowing linear transformations of the space and time variables. Over time, the process tends to drift towards its long-term mean: such a process is called mean-reverting.

SDE:

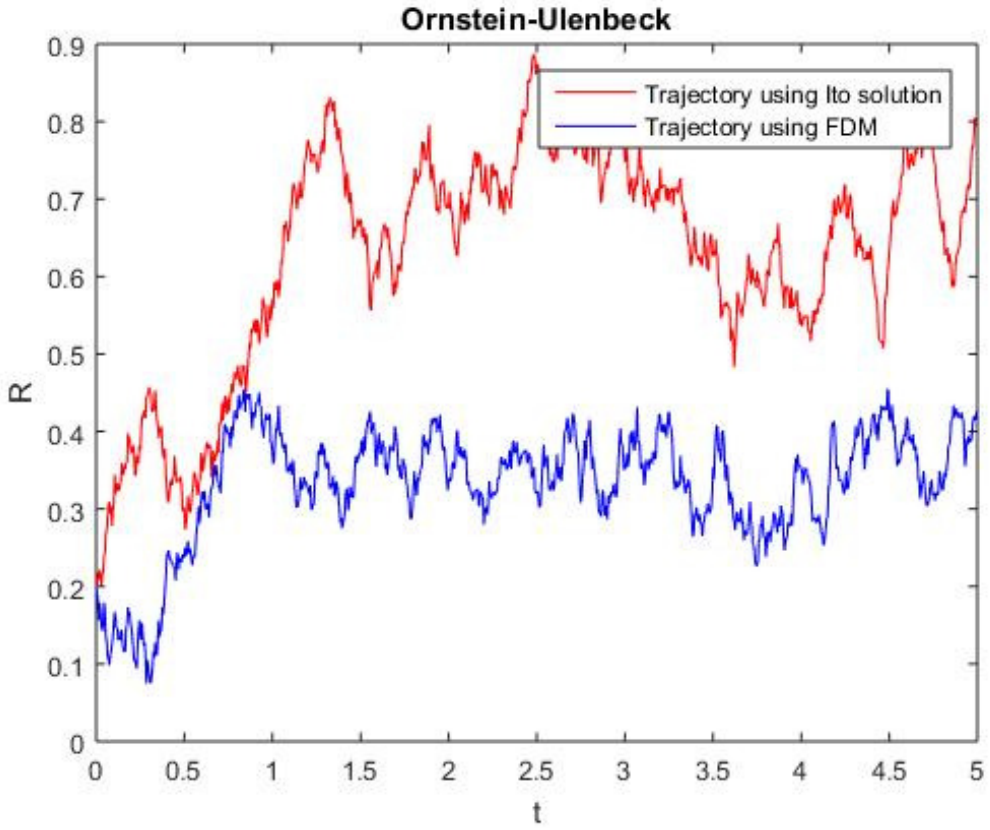
$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t$$

Analytical solution (Ito):

$$S_t = S_0 \exp(-\theta t) + \mu\left(1 - e^{-\theta t}\right) + \sigma \int_0^t e^{-\theta(t-s)} dW_s$$

Underlying functions: BMsimulator, Normal_polar

Initial data: R0 = 0.2; a = 0.3; b = 0.8; sigma = 0.2; T = 5; delta_t = 0.005;


Ornstein-Ulenbeck
Trajectory using Ito solution
Trajectory using FDM
R
t
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

ANNEXES

ANNEX 1 : Method of Polar Coordinates

```
function [alpha] = normal_polar(num_of_iter,mu,sigma)


%% Generating of U,Y - independent r.v
k = 1;
while k <= num_of_iter

      V1 = 2*rand()-1;
      V2 = 2*rand()-1;
      S = V1^2 + V2^2;

      if S < 1
         alpha(k) = V1*(-2*log(S)/S)^(1/2);
         alpha(k) = alpha(k)*sigma + mu;
         k = k+1;
      end;

end;

return;
```

ANNEX 2 : Reject method

```
function [alpha] = normal_reject(theta,num_of_iter,mu,sigma)

%% Generating of U,Y - independent r.v
k = 1;
while k <= num_of_iter

    Y(k) = log(abs(1-rand()))/(-theta); % Formula for exponential distribution
    u(k) = rand();

    if u(k)<=exp(-((Y(k)-1)^2)/2)
       if rand()<0.5
         alpha(k) = Y(k);
       else
         alpha(k) = -Y(k);
       end;

       Y(k) = Y(k)*sigma + mu;
       k = k+1;
    end;

end;

return;
```

ANNEX 3: TCL method

```
function[alpha] = normal_TCL(num_of_iter, mu, sigma)

%% Definition of loops' size

num_of_rand = 10000;

%% Main program

   for k = 1:num_of_iter

      Arithmeticmean_ubar = mean(rand(num_of_rand,1)); % Calculating of arithmetic mean
      alpha(k) = (Arithmeticmean_ubar-0.5)*(12*num_of_rand)^(0.5)*sigma + mu; %
Looking for alpha non-centered

   end;

return;
```

ANNEX 4: Simulator of Brownian Motion

```
function[W] = BMsimulator(T,discretization_num_t,method)


delta_t = T/discretization_num_t;
t(1) = 0;
for q = 2:discretization_num_t
   t(q) = t(q-1) + delta_t;
end;


if (strcmp(method,'TCL'))

   W(1) = 0;

   gaussian = normal_TCL(discretization_num_t,0,1);

   for i = 2:discretization_num_t

     W(i) = sqrt(delta_t)*sum(gaussian(1:i));
     % W(i) = sqrt(t(i))*sum(gaussian(1:i))/sqrt(i);

   end;

elseif (strcmp(method,'Reject'))

   W(1) = 0;

   gaussian = normal_reject(1,discretization_num_t,0,1);

   for i = 2:discretization_num_t
```

```
        % W(i) = sqrt(delta_t)*sum(gaussian(1:i));
        W(i) = sqrt(t(i))*sum(gaussian(1:i))/sqrt(i);

    end;

elseif (strcmp(method,'Polar'))

    W(1) = 0;

    gaussian = normal_polar(discretization_num_t,0,1);

    for i = 2:discretization_num_t

        W(i) = sqrt(delta_t)*sum(gaussian(1:i));
        % W(i) = sqrt(t(i))*sum(gaussian(1:i))/sqrt(i);

    end;

elseif (strcmp(method,'Random walk'))

    W(1) = 0;
    r = rand();
    x(1) = (r>0.5)*(-1)+(r<=0.5)*(1);
    for i = 2:discretization_num_t

        if rand()>0.5
            x(i) = -1;
        else
            x(i) = 1;
        end;

        W(i) = sqrt(t(i))*sum(x(1:i))/sqrt(i);
        % W(i) = sqrt(t(i))*sum(gaussian(1:i))/sqrt(i);
```

```
    end;

end;


return;
```

ANNEX 5: Geometric Brownian Motion

Using ITO exact solution (function returns a path)

```
function [S] = GBM_ITOSimulator(S0,mu,sigma,T,delta_t)

discretization_num_t = T/delta_t;

t(1) = 0;
for i = 2:discretization_num_t
    t(i) = t(i-1)+delta_t;
end;

W = BMsimulator(T,discretization_num_t,'Polar');
S(1) = S0;
for i = 2:discretization_num_t
    S(i) = S(1)*exp((mu-sigma^2/2)*t(i)+sigma*W(i));
end;

return;
```

Using FDM (function returns a path)

```
function [S] = GBM_FDMSimulator(S0,mu,sigma,T,delta_t)

discretization_num_t = T/delta_t;

t(1) = 0;
for i = 2:discretization_num_t
    t(i) = t(i-1)+delta_t;
end;

W = BMsimulator(T,discretization_num_t,'Polar');
S(1) = S0;
```

```
for i = 2:discretization_num_t
    S(i) = S(i-1) + S(i-1)*((mu-sigma^2/2)*delta_t+sigma*(W(i)-W(i-1)));
end;

return;
```

Proof of the concept

```
clear; clc;

S0 = 10;
mu = 0.1;
sigma = 0.5;
T = 0.5;
delta_t = 0.005;

discretization_num_t = T/delta_t;

t(1) = 0;
for i = 2:discretization_num_t
    t(i) = t(i-1)+delta_t;
end;



S = GBM_ITOSimulator(S0,mu,sigma,T,delta_t);
S2 = GBM_FDMSimulator(S0,mu,sigma,T,delta_t);

plot(t,S,'r');
hold on;
plot(t,S2,'b');
xlabel('t'); ylabel('S');
title('Geometric brownian motion')
legend('Trajectory using Ito solution', 'Trajectory using FDM');
```

ANNEX 6: Ornstein-Ulenbeck model

Using ITO exact solution (function returns a path)

```
function [R] = Ornstein_ITOSimulator(R0,a,b,sigma,T,delta_t)

discretization_num_t = T/delta_t;

t(1) = 0;
for i = 2:discretization_num_t
   t(i) = t(i-1)+delta_t;
end;

W = BMsimulator(T,discretization_num_t,'Polar');
R(1) = R0;
for i = 2:discretization_num_t

   for s = 2:i
      integral(s) = exp(a*(t(s)-t(i)))*(W(s)-W(s-1));
   end;
   R(i) = R(1)*exp(-a*t(i))+b*(1-exp(-a*t(i)))+sigma*sum(integral);
end;

return;
```

Using FDM (function returns a path)

```
function [R] = Ornstein_FDMSimulator(R0,a,b,sigma,T,delta_t)

discretization_num_t = T/delta_t;

t(1) = 0;
for i = 2:discretization_num_t
   t(i) = t(i-1)+delta_t;
```

```
end;

W = BMsimulator(T,discretization_num_t,'Polar');
R(1) = R0;
for i = 2:discretization_num_t
    R(i) = R(i-1) + a*(b-R(i-1))*delta_t + sigma*(W(i)-W(i-1));
end;

return;
```

Proof of the concept

```
clear; clc;

R0 = 0.2;
a = 0.3;
b = 0.8;
sigma = 0.2;
T = 5;
delta_t = 0.005;

discretization_num_t = T/delta_t;

t(1) = 0;
for i = 2:discretization_num_t
    t(i) = t(i-1)+delta_t;
end;

R = Ornstein_ITOSimulator(R0,a,b,sigma,T,delta_t);
R2 = Ornstein_FDMSimulator(R0,a,b,sigma,T,delta_t);

plot(t,R,'r');
hold on;
plot(t,R2,'b');
```

```
xlabel('t'); ylabel('R');
title('Ornstein-Ulenbeck')
legend('Trajectory using Ito solution', 'Trajectory using FDM');
```

Bibliography

1. “Monte Carlo Simulation”, last accessed January 30, 2017, http://www.palisade.com/risk/monte_carlo_simulation.asp
2. Manolessou M., “E.I.S.T.I. - Mathematics Department IFI - QFRM.M2 (2016-17) Introduction to Monte- Carlo Simulation Part I”, last updated November 23, 2016, http://arel.eisti.fr
3. WhatIs, “Polar Coordinates”, last accessed January 30, 2017, http://whatis.techtarget.com/definition/polar-coordinates