Practical 8 - Barrier options

Elena de Toledo Hernández (100452170), Julia González Ramos(100452185)

May 26, 2024

Abstract

The primary objective of this practical is to explore the numerical challenges involved in the stochastic simulation of path-dependent options, and to develop methods for addressing these challenges using the Brownian bridge and barrier shifting techniques. Specifically, we will focus on understanding the biases and errors associated with different timestep methods, estimating these biases, and evaluating the accuracy of the simulations.

Key goals include:

1. Deriving and solving the boundary value problem (BVP) for European down-and-out calls, comparing it with vanilla calls.
2. Implementing numerical simulations using the Euler-Maruyama method and examining the weak convergence rates for both Gaussian and non-Gaussian processes.
3. Utilizing advanced techniques like the Brownian bridge and barrier shifting to enhance simulation accuracy.
4. Analyzing various stochastic processes such as the Ornstein-Uhlenbeck process and Geometric Brownian Motion (GBM) under different conditions, including barrier presence.
5. Estimating survival probabilities in stochastic population models through a combination of numerical and semi-analytical methods.

By comparing different approaches and their computational efficiency, we aim to identify the most effective strategies for simulating path-dependent options in financial mathematics.

Exercise 1. Deriving the exact solution, $C_{d/o}$, of European down-and-out Calls. Assume T is the expiry, K the strike, B < K the barrier, and the underlying obeys GBM with drift r and volatility σ.

i) Write down the complete boundary value problem (BVP) for $C_{d/o}$: PDE, initial/terminal conditions (if any), boundary conditions (BCs) (if any). Sketch the domain of that BVP, $\Omega^B_{d/o}$.Explain succintly if and why the BVP is well posed, and what it means. Repeat everything for the vanilla Call (i.e. without barrier) of solution Cv. Briefly comment on the differences between the BVP for $C_{d/o}$ and for C_v.

0.0.1 European down-and-out call option

First, let's discuss why the BS formula can be applied to barrier option pricing. Because the hedging argument in a Vanilla BS equation relies on differential increments, and a differential increment will not take the value of a function across the barrier, a barrier option still obeys the BS equation.

Once this has been done, let us sketch the domain for the BVP.

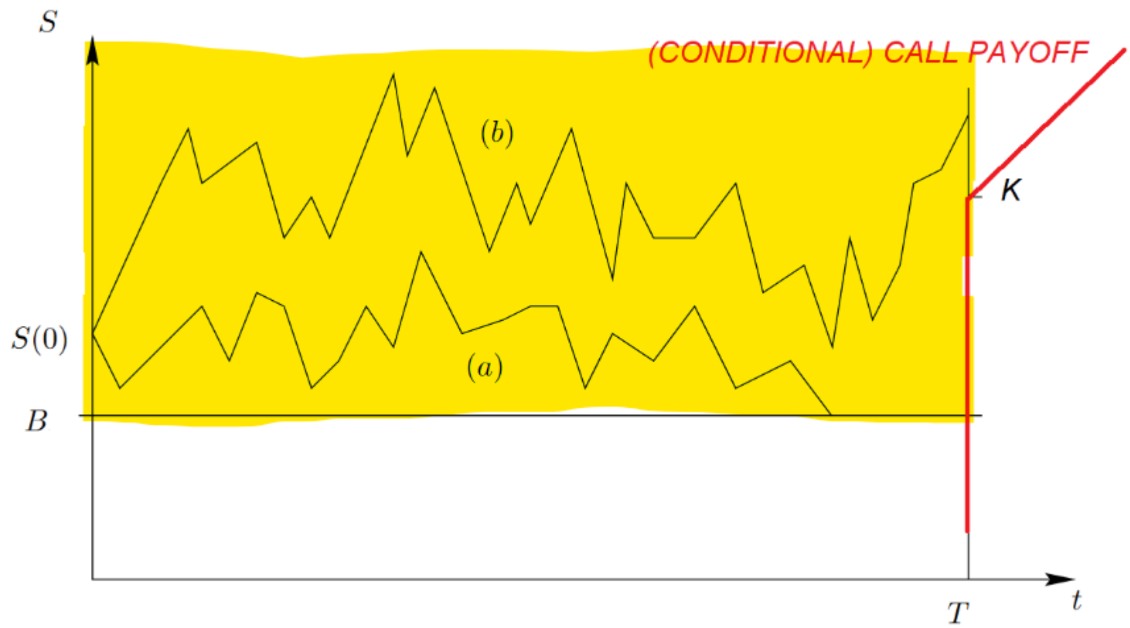

Figure 1: Domain

The domain is defined as $\Omega^B_{d/o} = (t, S) \in [0, T]x[B, \inf)$

Now, let's derive the complete BVP. Without boundary conditions, BS equation can have infinitely many solutions.

$$\frac{\partial C_{d/o}}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C_{d/o}}{\partial S^2} + rS\frac{\partial C_{d/o}}{\partial S} - rC_{d/o} = 0$$
$$C_{d/o}(T, S) = \max(S - K, 0)$$

Then, the boundary conditions of this domain are the following:

- At the barrier $S = B$:
$$C_{d/o}(t, B) = 0$$
- As $S \to \infty$:
$$C_{d/o}(t, S) \to S - Ke^{-r(T-t)}$$

For discussing whether the BVP is well posed, by researching different techniques we came out to find the method of images state below. Hence, we used the fact that we know the BVP of a European call option is well posed and derive our solution from this point.

1.-**First perform a change of variables**: We use the following transformations to convert the problem into a heat equation form:

$$S = Be^{-x}, \quad E = Be^{-k'}, \quad T = \frac{1}{4}\beta^2 t, \quad \frac{C_A}{B} = e^{-\alpha x + \alpha^2 \tau}u(x, \tau),$$

with

$$\alpha = \frac{1}{4}(1 - k'), \quad \beta = -\frac{1}{4}(k' - 1)^2, \quad k = \frac{\pi}{4\beta^2}, \quad k' = \frac{(r - D)t}{\beta^2}.$$

These transformations lead to the heat equation:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}, \quad \text{for } 0 < x < \infty, \ \tau > 0,$$

with initial and boundary conditions:

$$u(x,0) = u(T) = \max\left(e^{k'(1+1)x} - \left(\frac{E}{B}\right)e^{k'(1-1)x}, 0\right), \quad x \geq 0,$$

$$u(0,t) = 0.$$

2.-**Method of Images**: Next, we performed some research and came accross the method of images to handle the boundary condition $u(0,t) = 0$. By reflecting the initial data about $x = 0$ and changing its sign, the solution $u(x,\tau)$ for $-\infty < x < \infty$ satisfies:

$$u(x,\tau) = \begin{cases} U(x) & \text{for } x > 0, \\ -U(-x) & \text{for } x < 0, \end{cases}$$

where $U(x)$ is the solution for $x > 0$. This ensures the boundary condition $u(0,\tau) = 0$.

3.- **Solution for Initial Data**: The initial condition for $u(x,\tau)$ can then be written as:

$$u(x,0) = \begin{cases} \max\left(e^{k'(1+1)x} - \left(\frac{E}{B}\right)e^{k'(1-1)x}, 0\right) & \text{for } x > 0, \\ -\max\left(e^{k'(1+1)(-x)} - \left(\frac{E}{B}\right)e^{k'(1-1)(-x)}, 0\right) & \text{for } x < 0. \end{cases}$$

4.- **Ensuring Well-Posedness**: The heat equation on an infinite domain with given initial conditions and zero boundary conditions at the origin is a well-posed problem. Therefore, the transformed down-and-out call option problem is also well-posed.

5.- **Relating Back to the Original Problem**: The solution to the transformed problem correctly represents the solution to the original financial problem through the inverse of the variable transformations. Thus, the down-and-out call option is a well-posed boundary value problem.

Given that we are at a well posed BVP we know that the following three facts hold:
A solution exists.
The solution is unique.
The solution depends continuously on the initial and boundary conditions.

0.0.2 European vanilla call option

The BVP is easier for a European Vanilla Call Option. Let us denote the vanilla call by C_v.

Then, the BS formula for the vanilla call is.

$$\frac{\partial C_v}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C_v}{\partial S^2} + rS\frac{\partial C_v}{\partial S} - rC_v = 0$$
$$C_{d/o}(T,S) = \max(S-K,0)$$

Taking the BS equation stated above and evaluating it on the boundaries of S, we derive the following results.

$$\lim_{S\to 0^+} \frac{\sigma^2 S^2}{2}\frac{\partial^2 C_v}{\partial S^2} = 0 \quad \text{and} \quad \lim_{S\to 0^+} rS\frac{\partial C_v}{\partial S} = 0.$$

which when introduced into the BS formula yields this:

$$\frac{\partial C_v}{\partial t}(t,0) = rC_v(t,0) \quad \Rightarrow \quad C_v(t,0) = e^{-r(T-t)}C_v(T,0)$$

Hence, the boundaries for the European Call are:

$$C_v(t,0) = 0 \text{ and } \lim_{S\to+\infty} C_v(t,S) = S$$

0.0.3 Differences between the BVP's

The main difference is the additional boundary condition at S=B for the down-and-out call option, which causes the value of the option to drop to zero if the asset price hits the barrier B. Furthermore, the domain for the vanilla option include the below barrier part that is excluded in our down and out call.

ii) Prove that, if V (t, S) solves the Black-Scholes PDE (only the PDE, not the BVP!), then there is a unique value of γ such that the function $W(t,S) = S^\gamma V(t, H^2/S)$. Is there any restriction on the number H?

$$W(t,S) = S^\gamma V\left(t, \frac{H^2}{S}\right).$$

Let $z = \frac{H^2}{S}$. Then,

$$\frac{\partial z}{\partial S} = -\frac{H^2}{S^2} = -z\frac{1}{S}.$$

Compute the partial derivatives of $W(t,S)$:

$$\frac{\partial W}{\partial t} = S^\gamma \frac{\partial V}{\partial t}(t,z),$$

$$\frac{\partial W}{\partial S} = \gamma S^{\gamma-1}V(t,z) - S^\gamma \frac{z}{S}\frac{\partial V}{\partial z}(t,z) = \gamma S^{\gamma-1}V(t,z) - zS^{\gamma-1}\frac{\partial V}{\partial z}(t,z).$$

Compute the second order derivatives:

$$\frac{\partial^2 W}{\partial S^2} = \gamma(\gamma-1)S^{\gamma-2}V(t,z) - \gamma z S^{\gamma-2}\frac{\partial V}{\partial z}(t,z) - zS^{\gamma-2}\frac{\partial V}{\partial z}(t,z) - zS^{\gamma-2}\frac{\partial V}{\partial z}(t,z) - z^2 S^{\gamma-2}\frac{\partial^2 V}{\partial z^2}(t,z).$$

Substitute these into the Black-Scholes PDE for $W(t,S)$:

$$\frac{\partial W}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 W}{\partial S^2} + rS\frac{\partial W}{\partial S} - rW = 0.$$

Simplifying and rearranging terms gives a condition on γ:

$$\left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 z^2 \frac{\partial^2 V}{\partial z^2} + (r - \frac{\sigma^2}{2})z\frac{\partial V}{\partial z}\right) S^\gamma + \left(\frac{1}{2}\sigma^2\gamma(\gamma-1) + r\gamma - r\right) V = 0.$$

Setting the coefficient of V to zero for the equation to hold:

$$\frac{1}{2}\sigma^2\gamma(\gamma-1) + r\gamma - r = 0.$$

Solving this quadratic equation for γ yields:

$$\gamma = 0, \quad \gamma = 1 - \frac{2r}{\sigma^2}.$$

Hence, $\gamma = 0$ or $\gamma = 1 - \frac{2r}{\sigma^2}$ is the unique values such that $W(t,S) = SV(t, H^2/S)$ solves the Black-Scholes PDE.

Also derived from this result above, there's no restriction in the value of H, as long as it is a real number.

iii) Then show that $V(t,H) = W(t,H)/H^\gamma$.
Recalling the definition for $W(t,S)$:

$$W(t,S) = S^\gamma V\left(t, \frac{H^2}{S}\right).$$

First, we need to evaluate $W(t,S)$ at $S = H$, which is trivial given the equality above:

$$W(t,H) = H^\gamma V\left(t, \frac{H^2}{H}\right) = H^\gamma V(t,H).$$

We need to isolate $V(t,H)$. Therefore, divide both sides by H^γ:

$$V(t,H) = \frac{W(t,H)}{H^\gamma}.$$

Thus, we have shown that:

$$V(t,H) = \frac{W(t,H)}{H^\gamma}.$$

iv) Compute $\lim_{S\to+\infty} \frac{W(t,S)}{H^\gamma}$ and also $\lim_{S\to+\infty} \frac{W(T,S)}{H^\gamma}$(sketch it)

Recall the definition of $W(t,S)$:

$$W(t,S) = S^\gamma V\left(t, \frac{H^2}{S}\right).$$

First, consider the limit as $S \to +\infty$:

$$\lim_{S\to+\infty} \frac{W(t,S)}{H^\gamma} = \lim_{S\to+\infty} \frac{S^\gamma V\left(t, \frac{H^2}{S}\right)}{H^\gamma}.$$

Since $\frac{H^2}{S} \to 0$ as $S \to +\infty$, we need to consider the behavior of $V(t,z)$ as $z \to 0$. As discussed in the sections above, European call options, $V(t,z) \to 0$ as $z \to 0$. Thus,

$$\lim_{S\to+\infty} S^\gamma V\left(t, \frac{H^2}{S}\right) \to 0.$$

Therefore,

$$\lim_{S\to+\infty} \frac{W(t,S)}{H^\gamma} = 0.$$

Now, consider the limit at maturity T:

$$\lim_{S\to+\infty} \frac{W(T,S)}{H^\gamma} = \lim_{S\to+\infty} \frac{S^\gamma V\left(T, \frac{H^2}{S}\right)}{H^\gamma}.$$

At maturity T, for a European call option, $V(T,S)$ behaves like $\max(S-K,0)$. Therefore, as $S \to +\infty$,

$$V\left(T, \frac{H^2}{S}\right) \to 0,$$

because $\frac{H^2}{S}$ will be very small and likely out of the money.
Hence,

$$\lim_{S\to+\infty} \frac{W(T,S)}{H^\gamma} = 0.$$

Then, both limits result in zero:

$$\lim_{S \to +\infty} \frac{W(t,S)}{H^\gamma} = 0,$$

$$\lim_{S \to +\infty} \frac{W(T,S)}{H^\gamma} = 0.$$

Now, let's sketch these limits. As $S \to +\infty$, $W(t,S)/H^\gamma$ approaches zero.

v)What is the BVP that $W(t,S)$ solves in $\Omega^B_{d/o}$? Prove, then, that

$$C_{d/o}(t,S) = C_v(t,S) - \left(\frac{S}{B}\right)^{1-2r/\sigma^2} C_v\left(t, \frac{B^2}{S}\right).$$

The boundary value problem (BVP) that $W(t,S)$ solves in Ω_B can be described by the Black-Scholes partial differential equation (PDE):

$$\frac{\partial W}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 W}{\partial S^2} + rS\frac{\partial W}{\partial S} - rW = 0$$

with the boundary conditions:

$$\begin{cases} W(t,B) = 0, & \text{for } t \in [0,T] \\ W(T,S) = \max(S-K,0), & \text{for } S \in [B,\infty) \end{cases}$$

We prove that the down-and-out call option price $C_{\text{d.o.}}(t,S)$ is given by:

$$C_{\text{d.o.}}(t,S) = C_v(t,S) - \left(\frac{S}{B}\right)^{1-2r/\sigma^2} C_v(t,BS)$$

Here, $C_v(t,S)$ is the price of a European vanilla call option.
Proof
1. **Black-Scholes PDE Solution for C_v**:
The vanilla call option $C_v(t,S)$ satisfies the Black-Scholes PDE:

$$\frac{\partial C_v}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C_v}{\partial S^2} + rS\frac{\partial C_v}{\partial S} - rC_v = 0$$

2. **Method of Images**:
We use the method of images to construct the solution for the down-and-out call. Reflecting the stock price at the barrier and adjusting for the boundary conditions gives:

$$C_{\text{d.o.}}(t,S) = C_v(t,S) - \left(\frac{S}{B}\right)^{1-2r/\sigma^2} C_v(t,BS)$$

3. **Boundary and Initial Conditions**:
- At $S = B$, the down-and-out call option value is zero: $C_{\text{d.o.}}(t,B) = 0$. - As $S \to \infty$, the option behaves like a standard call option: $C_v(t,S) \to S - Ke^{-r(T-t)}$.
4. **Verification**:
Using the MATLAB functions 'blsprice' and 'barrierbybls', we can verify the formula by computing the prices for a European call and a down-and-out call and comparing them.
MATLAB Code

CODE 1: Verify Down-and-Out Call Option Pricing

```
% Parameters
S = 50;  % underlying price
K = 50;  % strike price
r = 0.035;  % risk-free rate
T = 1;  % time to maturity
sigma = 0.3;  % volatility
B = 45;  % barrier price
```

```matlab

% Compute the Black-Scholes price of a vanilla call option
Cv_bs = blsprice(S, K, r, T, sigma);

% Compute the price of a down-and-out call option using the formula
Cdo_theory = Cv_bs - (S/B)^(-2*r/sigma^2) * blsprice(B^2/S, K, r, T, sigma);


% Compute the price of a down-and-out call option using the barrier option pricing function
Settle = datetime(2015,1,1);
Maturity = datetime(2016,1,1);
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, 'EndDates', Maturity, ...
'Rates', r, 'Compounding', -1, 'Basis', 1);

StockSpec = stockspec(0.3, S);

Cdo = barrierbybls(RateSpec, StockSpec, 'call', K, Settle,...
Maturity,  'DO', B);

% Display the results
fprintf('Price of vanilla call option: %.4f\n', Cv_bs);
fprintf('Price of down-and-out call option (formula): %.4f\n', Cdo_theory);
fprintf('Price of down-and-out call option (barrier function): %.4f\n', Cdo);
```

This MATLAB code computes the down-and-out call option price using the derived formula and verifies it with the built-in ‘barrierbybls‘ function. These are the results:

- Price of vanilla call option: 6.7586
- Price of down-and-out call option (formula): 4.6615
- Price of down-and-out call option (barrier function): 4.4285

vi) Does the previous equality hold if the underlying sheds dividends? And if $r = r(t)$ (but deterministic)? And if $\sigma = \sigma(t)$ (deterministic)? And if S_t obeys Heston's model instead of GBM? *Hint: tread carefully here.*

The formula above is derived from a change of variables performed in the original formula by Black-Scholes. This original formula doesn't permit the shedding of dividends during the life of the option. In order to take this variable into account, we would need to perform the preceding analysis on the Black-Scholes formula for dividends:

$$\frac{\partial V}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} + (r-q) S \frac{\partial V}{\partial S} - rV = 0, \quad V(t=T, S) = \psi(S)$$

Regarding the use of a r=r(t) deterministic and $\sigma = \sigma(t)$ deterministic too, we think that making the proper adjustments, this would be a possibility. However, the model for the movement of the underlying would not longer be a Geometric Brownian Motion, but a modification of it using local volatility.

Finally, if S_t followed a Heston model, the formula would have to undergo numerous modifications, as the original Black-Scholes formula assumes that the underlying follows a Geometric Brownian Motion, not an Stochastic Volatility model such as the Heston model.

vii) Using only Matlab `blsprice` as blackbox, write a little Matlab script which displays the 3D solution of $C_{d/o}(t, S)$ for $K < B$ and $K > B$.

CODE 2: Down-and-Out Call Option Pricing

```matlab
% Parameters
K = 40; % Strike
B = 60; % Barrier
r = 0.05; % Risk-free rate
sigma = 0.2; % Volatility
T = 1; % Time to maturity
Smin = 0; % Minimum stock price
```

```
Smax = 200; % Maximum stock price -> just to put an end to the graph
tmin = 0; % Minimum time
tmax = T; % Maximum time

% Define the grid
N = 100; % Number of points in each direction
S = linspace(Smin, Smax, N);
t = linspace(tmin, tmax, N);
[S, t] = meshgrid(S, t);

% Compute the down-and-out call option price
Cdo = zeros(size(S));
for i = 1:N
    for j = 1:N
        % Time to maturity for each point
        tau = T - t(i, j);
        if S(i,j) < B
            Cdo(i,j) = 0; % The option is knocked out
        else
            % Vanilla call option price
            Cv_bs = blsprice(S(i,j), K, r, tau, sigma);
            % Down-and-out call option price
            Cdo(i,j) = Cv_bs - (S(i,j)/B)^(-2*r/sigma^2) * blsprice(B^2/S(i,j), K, r
                , tau, sigma);
        end
    end
end

% Plot the down-and-out call option price as a surface
surf(S, t, Cdo)
title('Down-and-Out Call Option Price')
xlabel('Stock Price')
ylabel('Time to Maturity')
zlabel('Option Price')
```

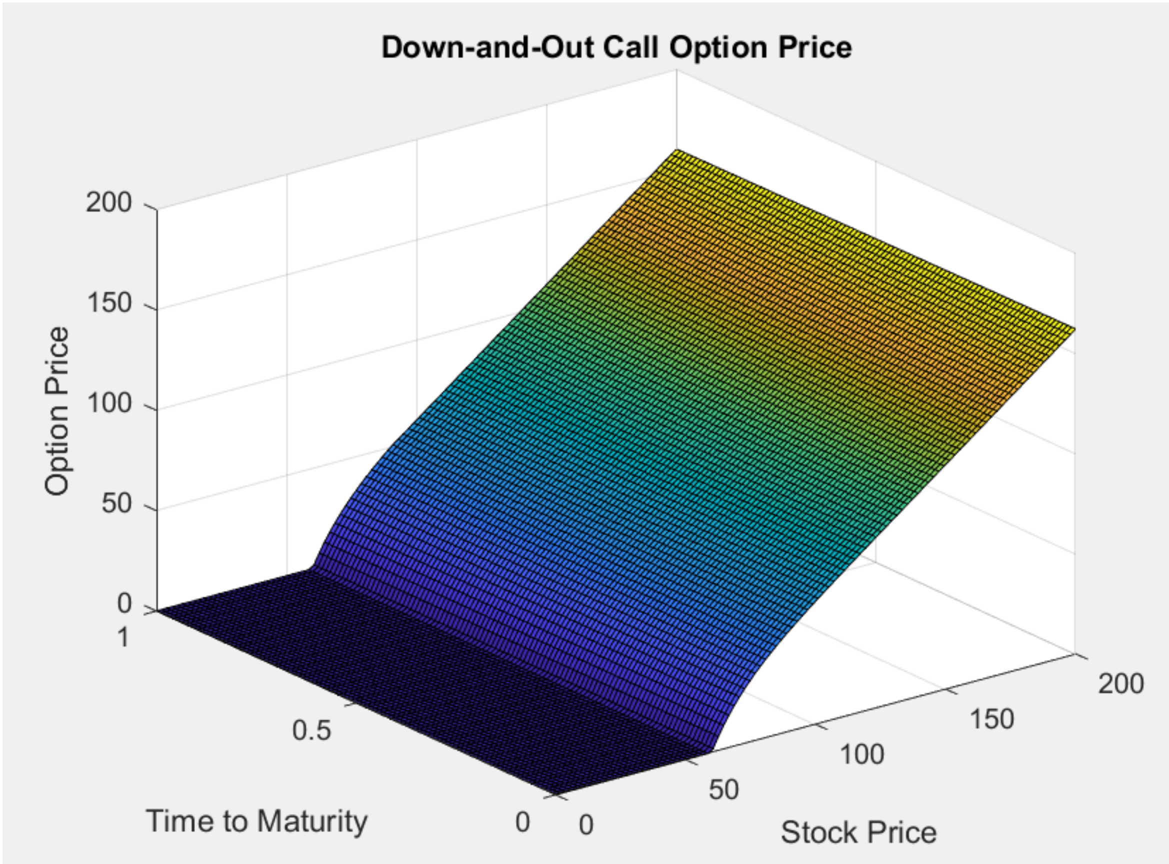

Figure 2: Case K B down and out Call

The value surface for a down-and-out barrier option with a strike price below the barrier level demonstrates a drop as the underlying asset approaches the barrier from above. This pronounced drop intensifies as the risk of the option being knocked out increases near the barrier.
However, when the asset's price moves above the barrier and away from the knockout risk, the option's value may rise, yet it remains lower than that of a comparable call option without a barrier, reflecting the persistent risk of knockout.

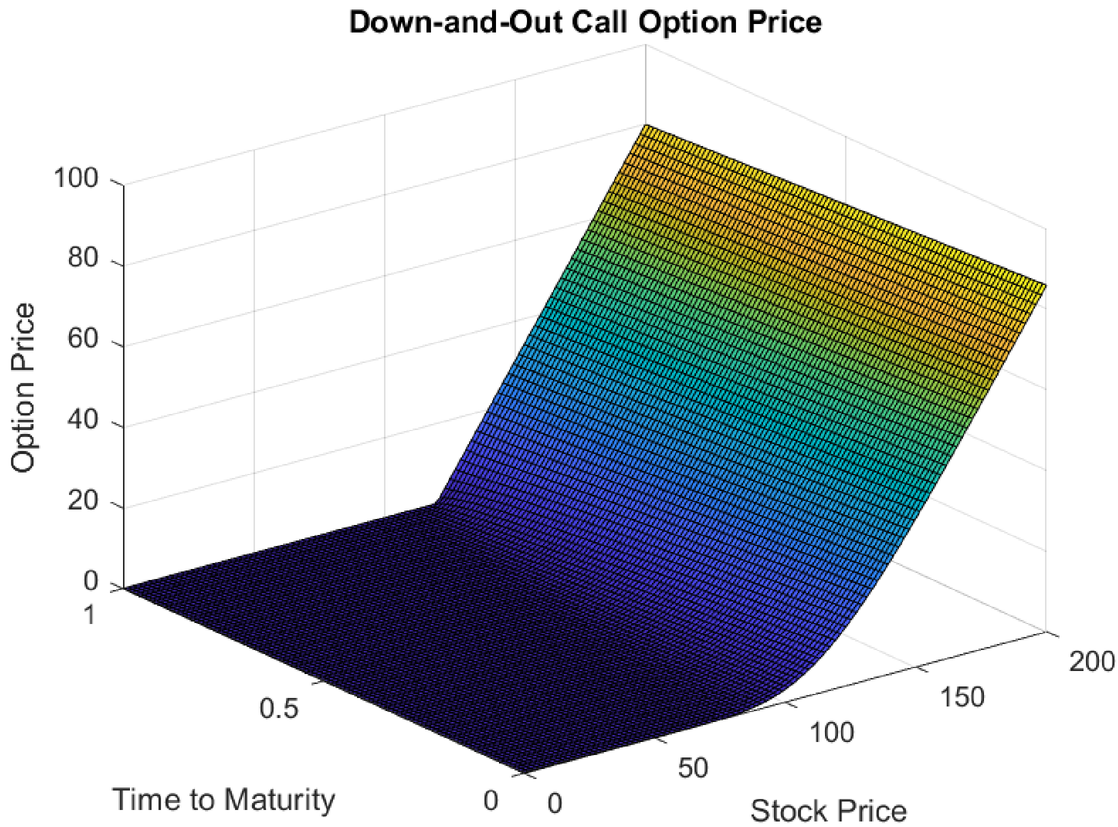

Figure 3: Case K   B down and out Call

The value surface of this down-and-out barrier option where the strike price is greater than the barrier shows a downward slope as time advances and the underlying asset's price nears the barrier from above, highlighting the growing risk of breaching the barrier due to time elapse and increased volatility.
Close to the barrier, the option's value sharply falls to zero, reflecting the high risk of the option being invalidated if the barrier is crossed. Conversely, when the price is far from and above the barrier, the option more closely resembles a standard call option, although its value remains sensitive to the barrier's proximity and the underlying's volatility.

Exercise 2. Consider an Ornstein-Uhlenbeck process $dS_t = \kappa(\theta - S_t)\,dt + \sigma\,dW_t$. The goal is to compute numerically $E[\psi]$, where $\psi = (S_T - K)_+$ if $\{\min_{0\leq t\leq T} S_t\} > B$, or $\psi = 0$ otherwise. Let $S_0 = 14$, $\theta = 14$, $\kappa = 2$, $\sigma = 0.5$ and $B = 13.5$, $K = 14$, $T = 2$. Start with the provided Matlab code `ouscratch.m`, and modify it to work on the several questions.

i) Choose N big enough that most of the error is bias and run `ouscratch.m` for several values of h. Since the OU process does not have an analytic solution, you can only estimate the error (now mostly bias) by the procedure indicated in the introduction. Plot the estimated error versus h on a loglog scale and estimate the weak convergence rate with and without the barrier. *Hint:* try with $N \geq 10^5$ and $h = T/M$ where $M = 8, 16, 32, \ldots$.

CODE 3: OU-scratch

```
function [V,ster,CPUt,varsc,eb]= ouscratch(N,M_,semilla,delta,varargin)
%[V,ster,CPUt,varsc,eb]= ouscratch(1e5,128,-1,0.5,true);

if semilla~=-1, rng(semilla); end
pinta= false; if nargin==5, pinta= true; end
[S0,T,K,B0,sigma,kappa,theta,r]= deal(14,2,14,13.5,.5,2,14,0);
Wfino= randn(N,2*M_);

for k=[1,2]
    tic
    if k==1, dW= (Wfino(:,1:2:end-1)+Wfino(:,2:2:end))/sqrt(2); else, dW=Wfino; end
    M= k*M_; h= T/M; S= NaN(N,M+1); S(:,1)= S0;

    B= B0; %no barrier shifting (DEFAULT)
    %B= B0+0.5826*sigma*sqrt(h); %upwards in this problem

    % First, run all trajectories at once without regard for barrier
    for j=1:M
        S(:,j+1)= S(:,j) + kappa*(theta-S(:,j))*h + sigma*sqrt(h)*dW(:,j);
    end

    % Second, detect barrier crossings
    HIT= ones(N,1); for j=1:N, if min(S(j,:))<=B, HIT(j)= 0; end; end

    % Third, compute payoffs
    score= NaN(N,1);
    score= exp(-r*T)*max(0,S(:,end)-K); %by default
    score(HIT==0)= 0; %overwritting trajectories which hit barrier

    % Finish
    V(k)= mean(score); %option price
    varsc(k)= var(score); ster(k)= 3*sqrt(varsc(k)/N); CPUt(k)= toc;
end
eb= (V(2)-V(1))/(1-2^delta);
if pinta
    for k=[1,2]
    fprintf('N=%d, h=%g: V=%g +/- %g CPUt=%g (%g%% hit barrier)\n',...
        N,T/(k*M_),V(k),ster(k),CPUt(k),100*(1-sum(HIT)/N))
    end
    fprintf('Estimated bias= %g\n',eb)
    if abs(eb)<2*max( [abs(ster(1)),abs(ster(2))] ) %N not big enough
        fprintf('WARNING: statistical error NOT negligible w.r.t. bias. Unreliable estimate!\n')
        eb= NaN;
    end
end
```

After running ouscratch with a large N and different M values here are the obtained results:

[V,ster,CPUt,varsc,eb]= ouscratch(1e5,8,-1,0.5,true);
N=100000, h=0.25: V=0.0994188 +/- 0.00155457 CPUt=0.0168272 (21.638% hit barrier)
N=100000, h=0.125: V=0.0938697 +/- 0.0014506 CPUt=0.0146248 (21.638% hit barrier)
Estimated bias= 0.0133968

[V,ster,CPUt,varsc,eb]= ouscratch(1e5,16,-1,0.5,true);
N=100000, h=0.125: V=0.0931927 +/- 0.00143631 CPUt=0.0277524 (22.647% hit barrier)
N=100000, h=0.0625: V=0.0894202 +/- 0.00138577 CPUt=0.0253393 (22.647% hit barrier)
Estimated bias= 0.00910772

[V,ster,CPUt,varsc,eb]= ouscratch(1e5,32,-1,0.5,true);
N=100000, h=0.0625: V=0.0890462 +/- 0.00138053 CPUt=0.0375836 (24.194% hit barrier)
N=100000, h=0.03125: V=0.0867821 +/- 0.00135533 CPUt=0.0460639 (24.194% hit barrier)
Estimated bias= 0.00546595

[V,ster,CPUt,varsc,eb]= ouscratch(1e5,64,-1,0.5,true);

N=100000, h=0.03125: V=0.0871614 +/- 0.00135651 CPUt=0.0803595 (25.671% hit barrier)
N=100000, h=0.015625: V=0.0854692 +/- 0.00134141 CPUt=0.0739414 (25.671% hit barrier)
Estimated bias= 0.00408536

[V,ster,CPUt,varsc,eb]= ouscratch(1e5,128,-1,0.5,true);
N=100000, h=0.015625: V=0.0852049 +/- 0.0013383 CPUt=0.139418 (27.025% hit barrier)
N=100000, h=0.0078125: V=0.0840996 +/- 0.00133025 CPUt=0.14406 (27.025% hit barrier)
Estimated bias= 0.00266835 WARNING: statistical error NOT negligible w.r.t. bias. Unreliable estimate!

As expected the output shows Based on the estimated bias, the weak error convergence follows sqrt(h).The inclusion of a barrier does not significantly impact the convergence rate, as both with and without the barrier, the error decreases proportionally to sqrt(h).

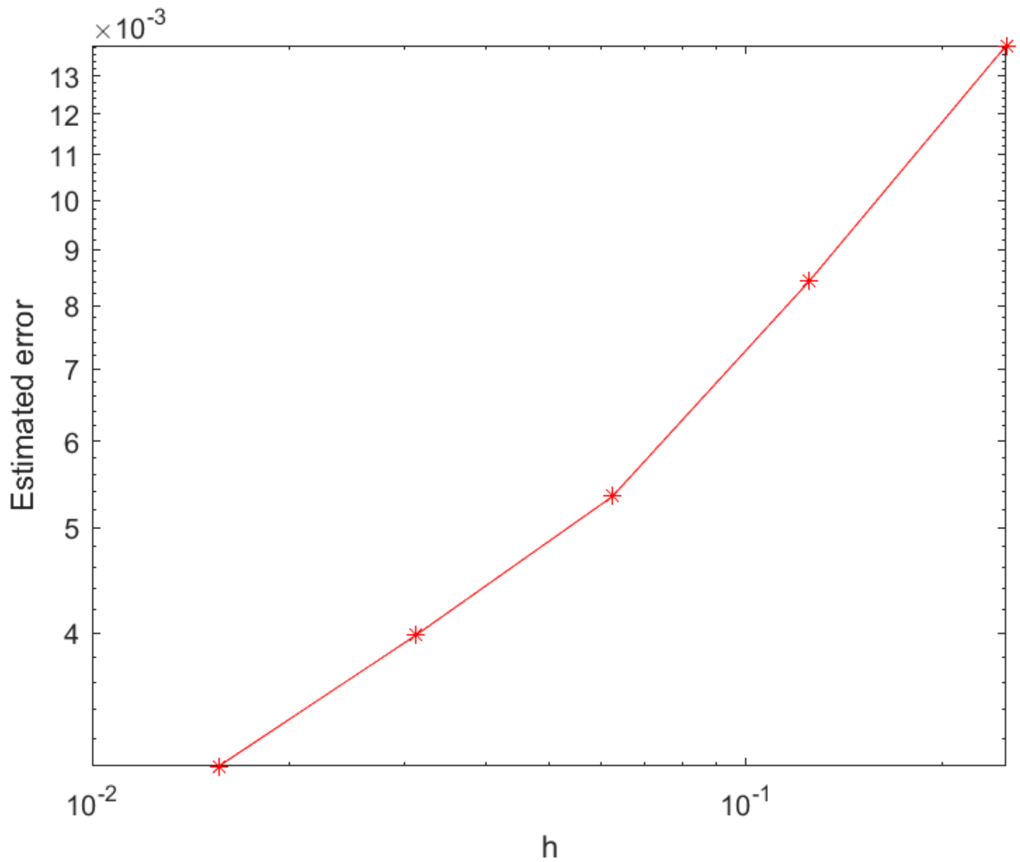

Figure 4: Estimated error vs h

ii) Now, use the Brownian bridge technique as explained in the lecture. (Recall that you must "freeze" the coefficients of the OU process within each timestep, and compute the "survival probability" within that step.) Estimate the bias again, then plot the convergence with h, and estimate δ again. *Hint:* now there is far less bias, so make it $N \geq 10^5$ (same h's).

CODE 4: OU-Bronwian Bridge

```
function [V, ster, CPUt, varsc, eb] = ouscratch_bb(N, M_, seed, delta, varargin)
if seed ~= -1
    rng(seed);
end
```

```

pinta = false;
if nargin == 5
    pinta = true;
end

[S0, T, K, B0, sigma, kappa, theta, r] = deal(14, 2, 14, 13.5, 0.5, 2, 14, 0);
Wfino = randn(N, 2 * M_);

for k = [1, 2]
    tic;
    if k == 1
        dW = (Wfino(:, 1:2:end-1) + Wfino(:, 2:2:end)) / sqrt(2);
    else
        dW = Wfino;
    end
    M = k * M_;
    h = T / M;
    S = NaN(N, M + 1);
    S(:, 1) = S0;

    B = B0; % No barrier shifting (DEFAULT)

    % First, run all trajectories at once without regard for barrier
    for j = 1:M
        mu = S(:, j) + kappa * (theta - S(:, j)) * h;
        sigma_sqrt_h = sigma * sqrt(h);

        % Generate next step using frozen coefficients
        S(:, j + 1) = mu + sigma_sqrt_h * dW(:, j);

        % Brownian bridge adjustment to check for barrier crossing within the step
        bridge = brownianBridge(N, h);  % Generate Brownian bridge for each step
        bridge_min = min(S(:, j), S(:, j + 1)) + bridge;

        % Adjust HIT based on Brownian bridge
        HIT = ones(N, 1);
        HIT(bridge_min <= B) = 0;
    end

    % Compute payoffs
    score = exp(-r * T) * max(0, S(:, end) - K); % By default
    score(HIT == 0) = 0; % Overwriting trajectories which hit barrier

    % Finish
    V(k) = mean(score); % Option price
    varsc(k) = var(score);
    ster(k) = 3 * sqrt(varsc(k) / N);
    CPUt(k) = toc;
end

eb = (V(2) - V(1)) / (1 - 2^delta);
if pinta
    for k = [1, 2]
        fprintf('N =%d, h =%g: V =%g +/- %g CPUt =%g (%g%% hit barrier)\n', ...
        N, T / (k * M_), V(k), ster(k), CPUt(k), 100 * (1 - sum(HIT) / N));
    end
    fprintf('Estimated bias = %g\n', eb);
    fprintf('abs(eb) = %g\n', abs(eb));
    fprintf('2 * max([abs(ster(1)), abs(ster(2))]) = %g\n', 2 * max([abs(ster(1)),
        abs(ster(2))]));

    if abs(eb) < 2 * max([abs(ster(1)), abs(ster(2))]) % N not big enough
        fprintf('WARNING: statistical error NOT negligible w.r.t. bias. Unreliable
            estimate!\n');
        eb = NaN;
    end
end
end

% Here is the modified brownianBridge function, integrated for stepwise barrier
```

```
    checking
function bridge = brownianBridge(N, T)
    bridge = zeros(N, 1);
    for i = 1:N
        % Generate each Brownian increment
        if i == 1
            bridge(i) = sqrt(T) * randn;
        else
            dt = T / N;  % equal time steps
            bridge(i) = bridge(i-1) + sqrt(dt) * randn;
        end
    end
end
```

CODE 5: Bronwian Bridge

```
function bridge = brownianBridge(N, T)
bridge = zeros(N, 1);
t = linspace(0, T, N+2)';
t = t(2:end-1);
bridge(1) = sqrt(T) * randn;
for i = 2:N
    dt = t(i) - t(i-1);
    bridge(i) = bridge(i-1) + sqrt(dt) * randn;
end
end
```

= ouscratch(1e6,8,-1,1,true);
N=1000000, h=0.25: V=0.11553 +/- 0.000506266 CPUt=1.14842 (0.7533% hit barrier)
N=1000000, h=0.125: V=0.106971 +/- 0.000468694 CPUt=1.69095 (0.7533% hit barrier)
Estimated bias= 0.00855905
abs(eb) = 0.00855905

V,ster,CPUt,varsc,eb
= ouscratch(1e6,16,-1,1,true);
N=1000000, h=0.125: V=0.106795 +/- 0.000468867 CPUt=1.06831 (0.8307% hit barrier)
N=1000000, h=0.0625: V=0.1032 +/- 0.000453089 CPUt=2.77341 (0.8307% hit barrier)
Estimated bias= 0.00359586
abs(eb) = 0.00359586

V,ster,CPUt,varsc,eb
= ouscratch(1e6,32,-1,1,true);
N=1000000, h=0.0625: V=0.103004 +/- 0.000452158 CPUt=2.26416 (0.9899% hit barrier)
N=1000000, h=0.03125: V=0.101326 +/- 0.000444843 CPUt=7.04694 (0.9899% hit barrier)
Estimated bias= 0.00167782
abs(eb) = 0.00167782

As it can be checked by analyzing the results compared to part (i), the bias obtain is considerably lower.

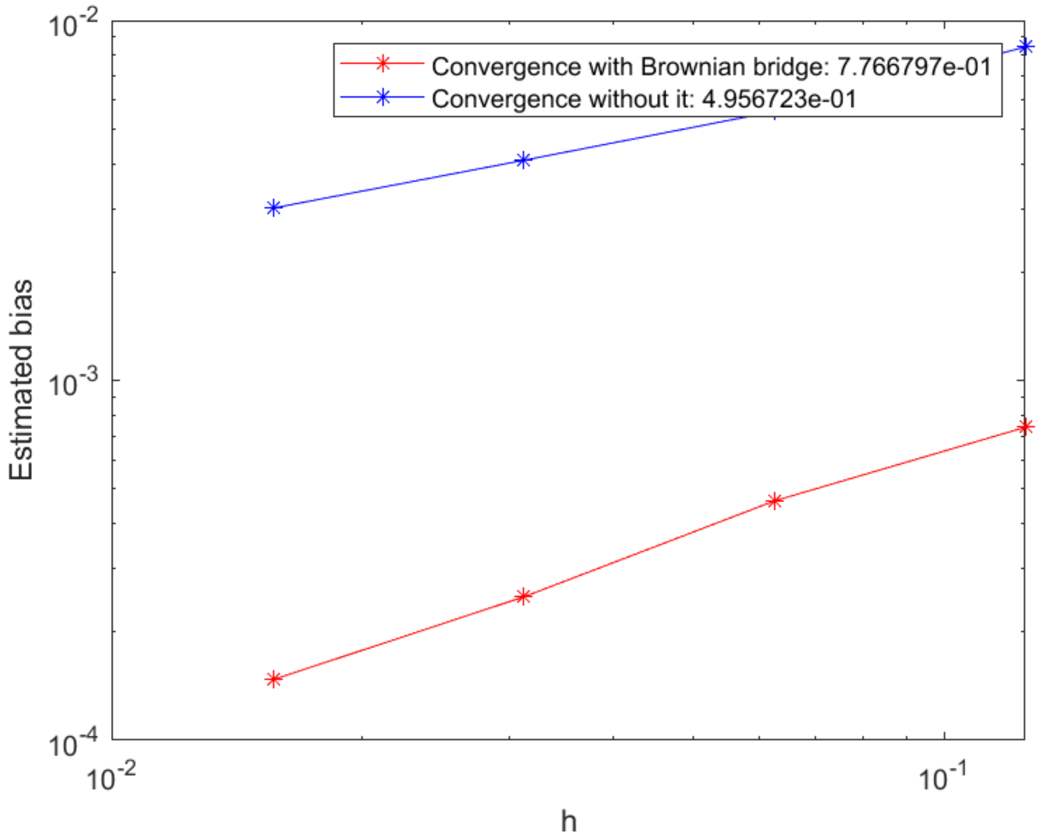

Figure 5: Estimated bias with Brownian Bridge OU

iii) Finally, use the "barrier shifting" strategy: the barrier must simply be shifted by $0.5826\sigma\sqrt{h}$. Discuss whether you need to raise the barrier or lower it in this example, and why. Redo the convergence analysis. *Hint:* simply comment/uncomment lines 14/15 of `ouscratch.m`.

The barrier must be shifted upwards. The reason is that we want to be more "strict" than the actual problem to ensure that we don't miss with this approximation to the real OU process any excursions the process may make.

CODE 6: OU-shifted

```
function [V,ster,CPUt,varsc,eb] = ouscratch_shifted(N, M_, semilla, delta, varargin)
    if semilla ~= -1
        rng(semilla);
    end
    pinta = false;
    if nargin == 5
        pinta = true;
    end
    [S0, T, K, B0, sigma, kappa, theta, r] = deal(14, 2, 14, 13.5, .5, 2, 14, 0);
    Wfino = randn(N, 2 * M_);

    for k = [1, 2]
        tic;
        if k == 1
            dW = (Wfino(:, 1:2:end-1) + Wfino(:, 2:2:end)) / sqrt(2);
        else
            dW = Wfino;
        end
```

```
        M = k * M_;
        h = T / M;
        S = NaN(N, M + 1);
        S(:, 1) = S0;

        B = B0; % No barrier shifting (DEFAULT)
        B = B0 + 0.5826 * sigma * sqrt(h); % Shift barrier upwards

        % First, run all trajectories at once without regard for barrier
        for j = 1:M
            S(:, j + 1) = S(:, j) + kappa * (theta - S(:, j)) * h + sigma * sqrt(h)
                * dW(:, j);
        end

        % Second, detect barrier crossings
        HIT = ones(N, 1);
        for j = 1:N
            if min(S(j, :)) <= B
                HIT(j) = 0;
            end
        end

        % Third, compute payoffs
        score = NaN(N, 1);
        score = exp(-r * T) * max(0, S(:, end) - K); % By default
        score(HIT == 0) = 0; % Overwriting trajectories which hit barrier

        % Finish
        V(k) = mean(score); % Option price
        varsc(k) = var(score);
        ster(k) = 3 * sqrt(varsc(k) / N);
        CPUt(k) = toc;
    end
    eb = (V(2) - V(1)) / (1 - 2^delta);
    if pinta
        for k = [1, 2]
            fprintf('N=%d, h=%g: V=%g +/- %g CPUt=%g (%g%% hit barrier)\n', ...
                N, T / (k * M_), V(k), ster(k), CPUt(k), 100 * (1 - sum(HIT) / N));
        end
        fprintf('Estimated bias= %g\n', eb);

        % Debug prints
        fprintf('abs(eb) = %g\n', abs(eb));
        fprintf('2 * max([abs(ster(1)), abs(ster(2))]) = %g\n', 2 * max([abs(ster(1)
            ), abs(ster(2))]));

        if abs(eb) < 2 * max([abs(ster(1)), abs(ster(2))]) % N not big enough
            fprintf('WARNING: statistical error NOT negligible w.r.t. bias.
                Unreliable estimate!\n');
            eb = NaN;
        end
    end
end
```

= ouscratch(1e6,8,-1,1,true);
N=1000000, h=0.25: V=0.0753201 +/- 0.000446631 CPUt=1.04187 (39.9207% hit barrier)
N=1000000, h=0.125: V=0.0785001 +/- 0.000431339 CPUt=0.928667 (39.9207% hit barrier)
Estimated bias= -0.00318003

V,ster,CPUt,varsc,eb
= ouscratch(1e6,16,-1,1,true);
N=1000000, h=0.125: V=0.0788248 +/- 0.000432596 CPUt=0.823742 (35.7474% hit barrier)
N=1000000, h=0.0625: V=0.0801804 +/- 0.000424843 CPUt=0.812546 (35.7474% hit barrier)
Estimated bias= -0.00135561

V,ster,CPUt,varsc,eb
= ouscratch(1e6,32,-1,1,true);
N=1000000, h=0.0625: V=0.0799446 +/- 0.0004246 CPUt=0.820203 (33.5816% hit barrier)

N=1000000, h=0.03125: V=0.0805801 +/- 0.000420794 CPUt=1.19541 (33.5816% hit barrier)
Estimated bias= -0.000635523

With barrier shifting, the weak error convergence improves to O(h).
Estimated weak convergence rate

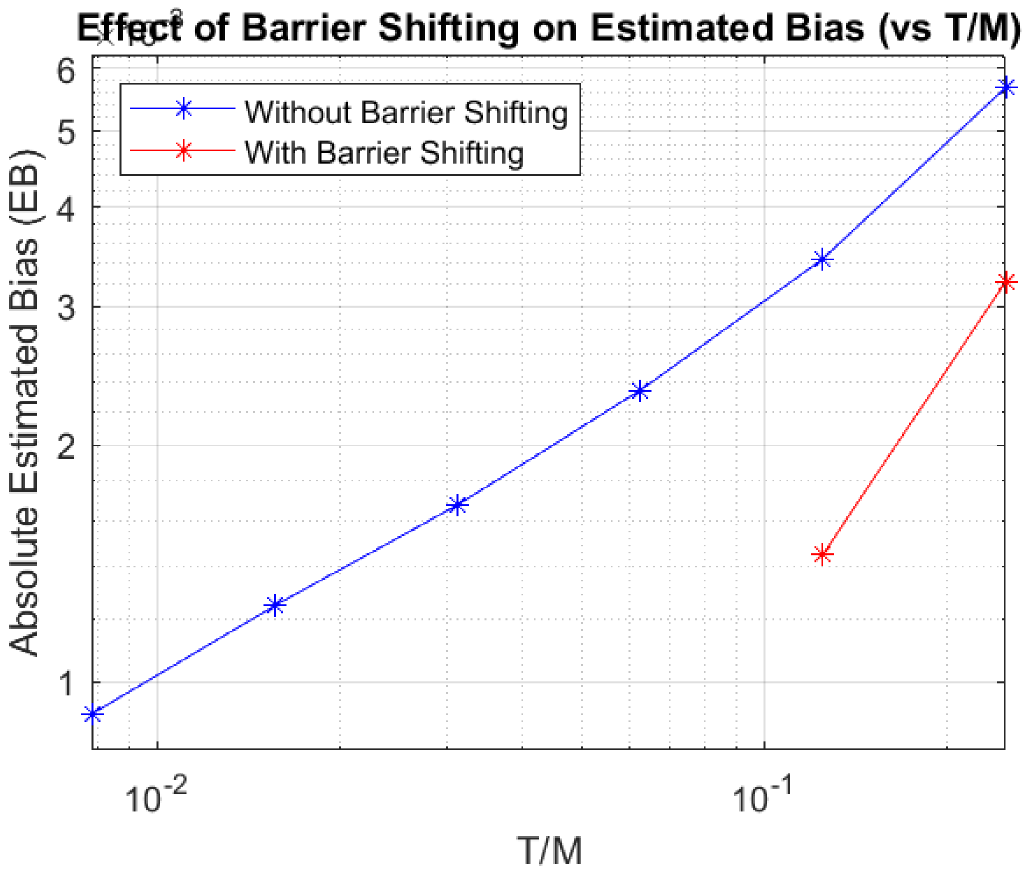

Figure 6: Barrier shifting estimated bias

Exercise 3. Let's see how results are much better when the underlying is a GBM. Specifically, set $S_0 = 100$, $K = 100$, $\sigma = 0.3$, $r = 0.1$, $T = 0.2$, $B = 85$.

i) Prove that the *exact* solution is $C_{d/o} = 6.3076$.

CODE 7: exact solution call d/o

```
% Prove that the exact solution is Cd/o = 6.3076.

S0 = 100; % Initial stock price
K = 100;  % Strike price
sigma = 0.3; % Volatility
r = 0.1;  % Risk-free interest rate
T = 0.2;  % Time to maturity
B = 85;   % Barrier level

C = blsprice(S0, K, r, T, sigma);
C_2 = blsprice(B^2/S0, K, r, T, sigma);

```

```
% Calculate the down-and-out call value
Cd_o = C - (S0/B)^(1-(2*r/sigma^2))*C_2;

fprintf('Exact solution for the down-and-out call option: %.4f \n',Cd_o);
```

Exact solution for the down-and-out call option: 6.3076

ii) Solve the GBM numerically "without tricks" and study weak convergence. clear all; close all randn('state',0)

CODE 8: GBM

```
% problem parameters and exact solution
r = 0.1; sig = 0.3; T = 0.2; S0 = 100; K = 100; B = 85; Ve= 6.3076;

% Monte Carlo simulation comparing to both
M = 1e+7;
M2 = 1e+4;
for p = 1:7
    N = 2^p; h = T/N; sum1 = 0; sum2 = 0; sum3 = 0; sum4 = 0;
    for m = 1:M2:M
        m2 = min(M2,M-m+1); S = S0*ones(1,m2); S2 = S0*ones(1,m2);
        for n = 1:N/2
            dW1 = sqrt(h)*randn(1,m2);
            S = S.*(1+r*h+sig*dW1);
            dW2 = sqrt(h)*randn(1,m2);
            S = S.*(1+r*h+sig*dW2);
            S2 = S2.*(1+r*2*h+sig*(dW1+dW2));
        end
        P = exp(-r*T)*max(S-K,0);
        P2 = exp(-r*T)*max(S2-K,0);

        sum1= sum1 + sum(P);
        sum2= sum2 + sum(P.^2);
        sum3= sum3 + sum(P-P2);
        sum4= sum4 + sum((P-P2).^2);
    end
    hh(p)= h;
    err1(p) = sum1/M - Ve;
    err2(p)= 3*sqrt((sum2/M - (sum1/M)^2)/(M-1));
    err3(p)= sum3/M;
    err4(p)= 3*sqrt((sum4/M - (sum3/M)^2)/(M-1));
end
figure; pos=get(gcf,'pos'); pos(3:4)=pos(3:4).*[0.8 0.8]; set(gcf,'pos',pos); loglog
    (hh,abs(err1),'b-*',hh,err2,'r-*'); title('Weak convergence -- comparison to
    exact solution');
xlabel('h'); ylabel('Error');
legend(' Weak error',' MC error','location', 'NorthWest')
figure; pos=get(gcf,'pos'); os(3:4)=pos(3:4).*[0.8 0.8]; set(gcf,'pos',pos);
loglog(hh,abs(err3),'b-*',hh,err4,'r-*');title('Weak convergence -- difference from
    2h approximation');
xlabel('h'); ylabel('Error');legend(' Weak error',' MC error','location','NorthWest'
    );
```

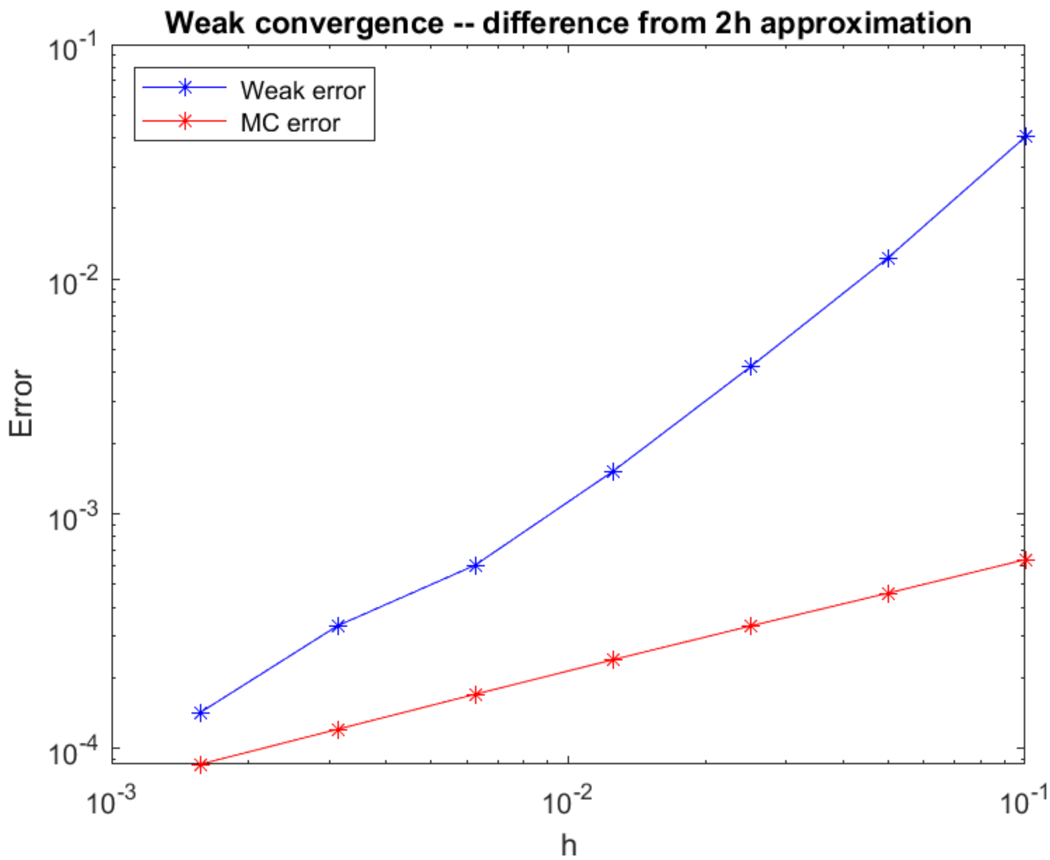

Figure 7: GBM weak convergence

As it can be observed in the above plot the weak convergence of GBM grows almost linearly as h increases. In the next iteration different techniques will be applied to reduce the error.

iii) Find the SDE for $\log S_t$ (instead of S_t) and repeat ii). Henceforth in the exercise (and in your life), always use the SDE for $\log S_t$ when handling GBM! *Hint: for this Exercise you can use/modify* `gbmscratch.m`*, which also is written to be light on memory, and so to allow many more trajectories. Note however that beyond $N > 10^7$ the random number generator may start being unreliable.*

CODE 9: log GBM

```
% problem parameters and exact solution
r = 0.1; sigma = 0.3; T = 0.2; S0 = 100; K = 100; B = 85; Ve = 6.3076;

% Monte Carlo simulation comparing to both
M = 1e7;
M2 = 1e4;
for p = 1:7
    N = 2^p; h = T / N; sum1 = 0; sum2 = 0; sum3 = 0; sum4 = 0;
    for m = 1:M2:M
        m2 = min(M2, M - m + 1); logS = log(S0) * ones(1, m2); logS2 = log(S0) *
            ones(1, m2);
        for n = 1:N/2
            dW1 = sqrt(h) * randn(1, m2);
            logS = logS + (r - 0.5 * sigma^2) * h + sigma * dW1;
            dW2 = sqrt(h) * randn(1, m2);
            logS = logS + (r - 0.5 * sigma^2) * h + sigma * dW2;
            logS2 = logS2 + (r - 0.5 * sigma^2) * 2 * h + sigma * (dW1 + dW2);
        end
```

```
        S = exp(logS);
        S2 = exp(logS2);
        P = exp(-r * T) * max(S - K, 0);
        P2 = exp(-r * T) * max(S2 - K, 0);

        sum1 = sum1 + sum(P);
        sum2 = sum2 + sum(P.^2);
        sum3 = sum3 + sum(P - P2);
        sum4 = sum4 + sum((P - P2).^2);
    end
    hh(p) = h;
    err1(p) = sum1 / M - Ve;
    err2(p) = 3 * sqrt((sum2 / M - (sum1 / M)^2) / (M - 1));
    err3(p) = sum3 / M;
    err4(p) = 3 * sqrt((sum4 / M - (sum3 / M)^2) / (M - 1));
end

% Plotting results for weak convergence
figure; pos = get(gcf, 'pos'); pos(3:4) = pos(3:4) .* [0.8 0.8]; set(gcf, 'pos', pos);
loglog(hh, abs(err1), 'b-*', hh, err2, 'r-*');
title('Weak convergence -- comparison to exact solution');
xlabel('h'); ylabel('Error');
legend('Weak error', 'MC error', 'location', 'NorthWest');

figure; pos = get(gcf, 'pos'); pos(3:4) = pos(3:4) .* [0.8 0.8]; set(gcf, 'pos', pos);
loglog(hh, abs(err3), 'b-*', hh, err4, 'r-*');
title('Weak convergence -- difference from 2h approximation');
xlabel('h'); ylabel('Error');
legend('Weak error', 'MC error', 'location', 'NorthWest');
```

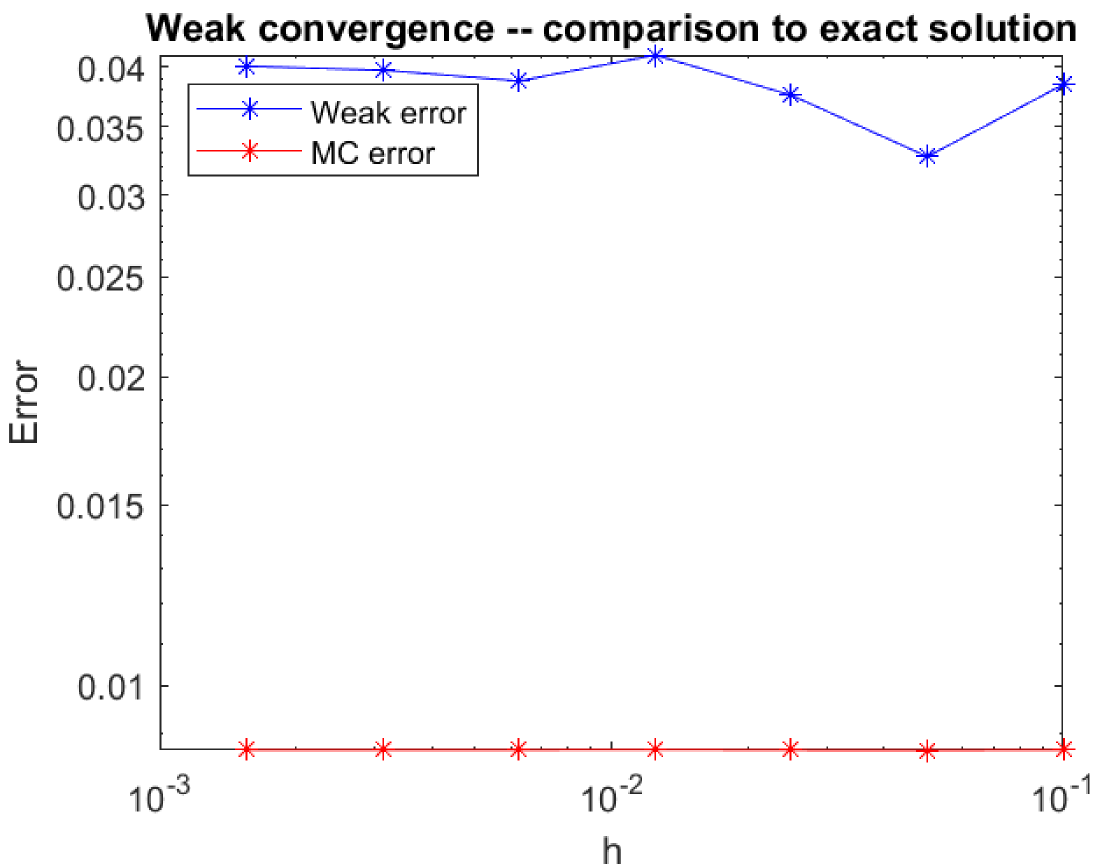

Figure 8: log_GBM weak convergence exact sol

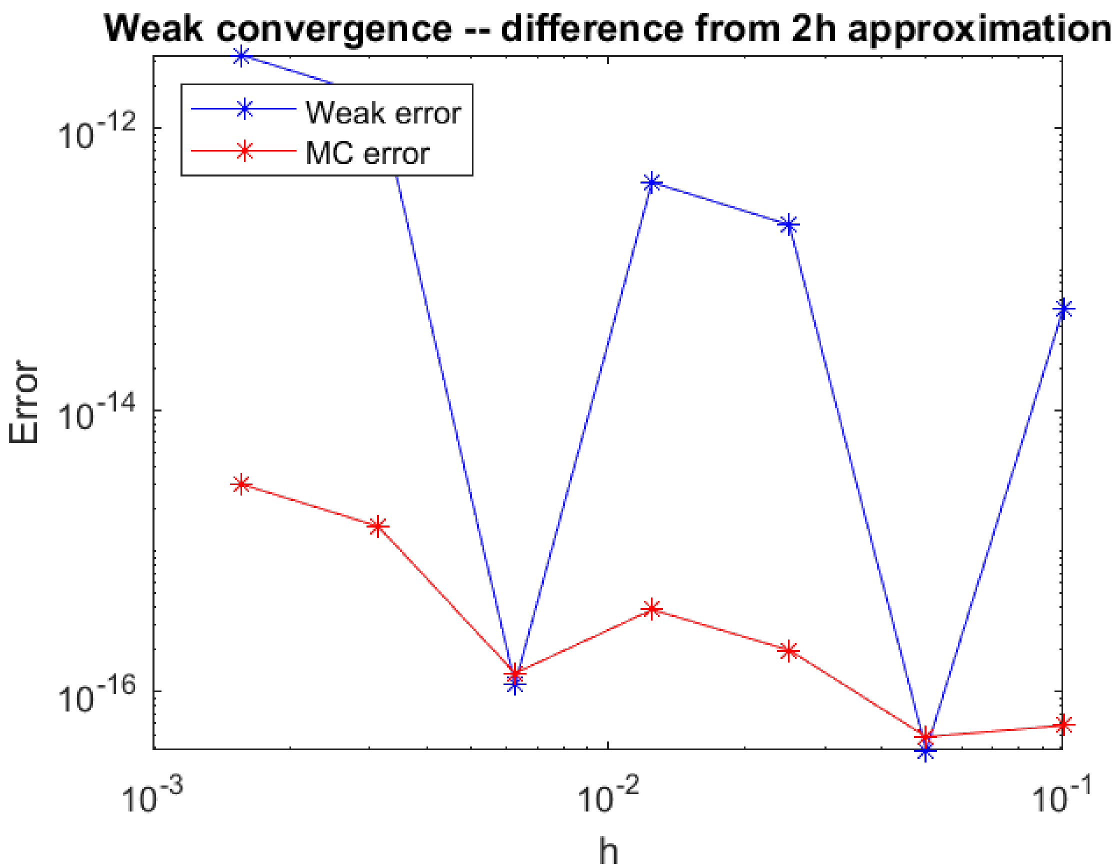

Figure 9: log_GBM weak convergence 2h

Transforming the stock price process to log(St) stabilizes numerical simulations and reduces the variance of the estimator. This logarithmic transformation is theoretically significant as it aligns with the log-normal distribution assumption of stock prices in the Black-Scholes model, facilitating easier, checked plotting the CPU time on section 3.7 and more accurate computation as can be observed by comparing this weak convergence plot compared to the one on the previous section, checking how the weak error stabilize and stops increasing linearly as in Euler-Maruyama.

iv) Shift the barrier, solve numerically, and study weak convergence again.

CODE 10: Barrier shiftting in GBM

```
% problem parameters and exact solution
r = 0.1; sigma = 0.3; T = 0.2; S0 = 100; K = 100;
barriers = [90, 80]; % Array of barrier levels
Ve = 6.3076; % This might need adjustment based on new barriers

% Monte Carlo simulation comparing to both
M = 1e7;
M2 = 1e4;

results = struct(); % To store results for different barriers

for b = 1:length(barriers)
    B = barriers(b);
    sum1 = zeros(1,7); sum2 = zeros(1,7);
    hh = zeros(1,7); err1 = zeros(1,7); err2 = zeros(1,7);

    for p = 1:7
        N = 2^p; h = T / N;
```

```
        sumV = 0; sumV2 = 0;
        for m = 1:M2:M
            m2 = min(M2, M - m + 1);
            logS = log(S0) * ones(1, m2);
            for n = 1:N
                dW = sqrt(h) * randn(1, m2);
                logS = logS + (r - 0.5 * sigma^2) * h + sigma * dW;
            end
            S = exp(logS);
            % Option only has value if it has never hit the barrier
            validPaths = all(S > B, 1);
            P = exp(-r * T) * max(S(end,:) - K, 0) .* validPaths;

            sumV = sumV + sum(P);
            sumV2 = sumV2 + sum(P.^2);
        end
        hh(p) = h;
        sum1(p) = sumV / M;
        sum2(p) = sumV2 / M;
        err1(p) = sum1(p) - Ve;
        err2(p) = 3 * sqrt((sum2(p) / M - (sum1(p) / M)^2) / (M - 1));
    end

    results(b).barrier = B;
    results(b).hh = hh;
    results(b).err1 = abs(err1);
    results(b).err2 = err2;

    % Plotting results for weak convergence for each barrier
    figure; loglog(hh, err1, 'b-*', hh, err2, 'r-*');
    title(['Weak convergence with Barrier = ', num2str(B)]);
    xlabel('h'); ylabel('Error');
    legend('Absolute Weak error', '3*STD MC error', 'location', 'NorthWest');
end
```

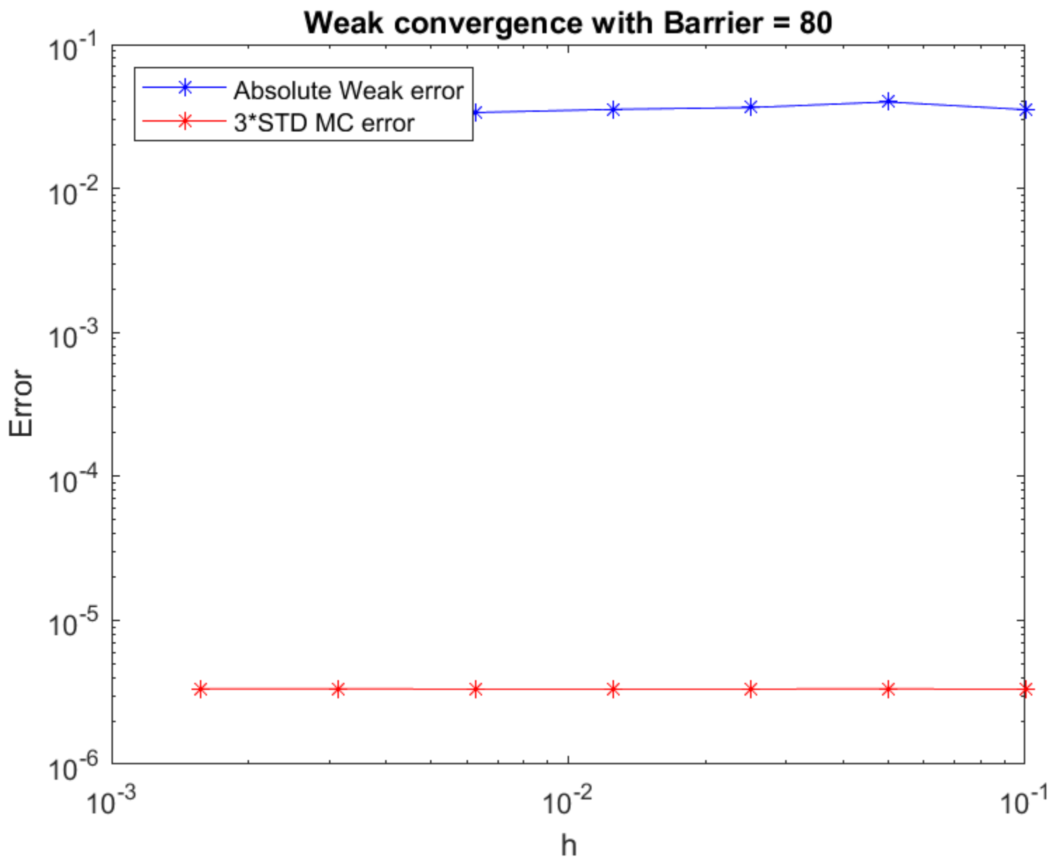

Figure 10: Weak Convergence with Barrier at 80

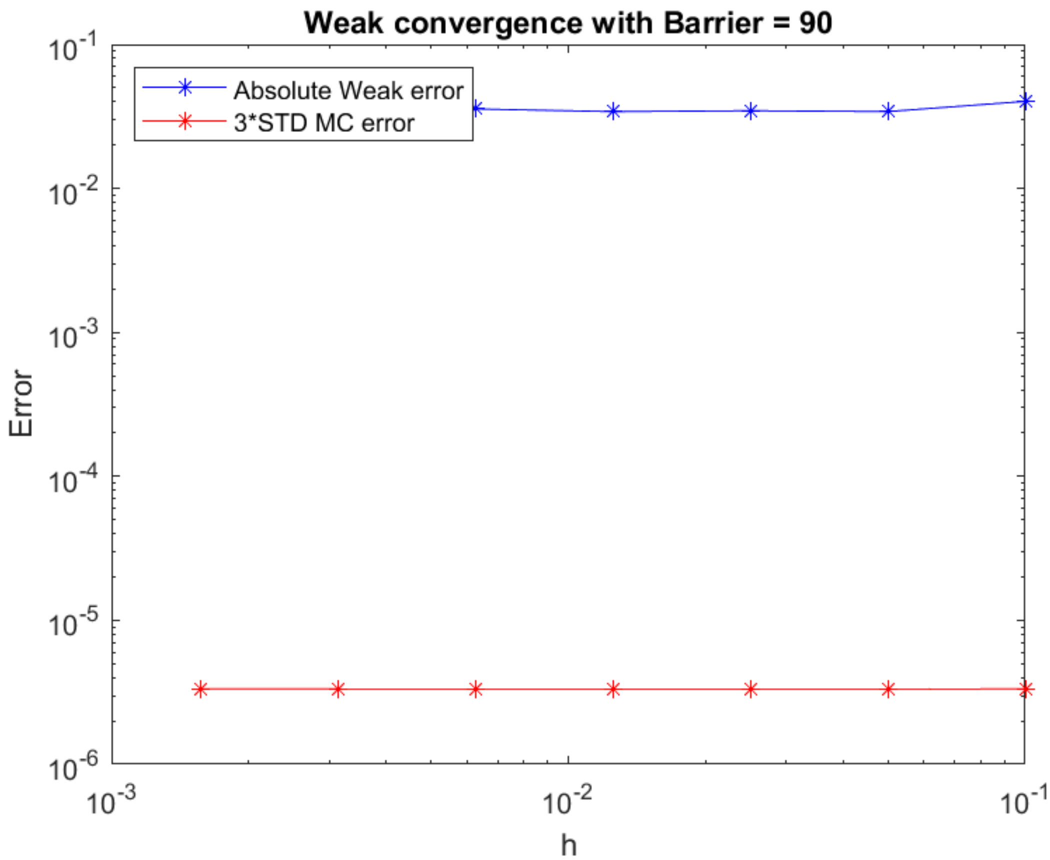

Figure 11: Weak Convergence with Barrier at 90

Theoretically, shifting barriers result in better approximating the continuous model, ensuring that the barrier is effectively captured within the discrete time steps of the simulation.

v) The same, but with a Brownian bridge at each timestep.

CODE 11: Brownian Bridge in GBM

```
% problem parameters
r = 0.1; sigma = 0.3; T = 0.2; S0 = 100; K = 100; Ve = 6.3076;

% Monte Carlo simulation setup
M = 1e7;
M2 = 1e4;
for p = 1:7
    N = 2^p; h = T / N; sum1 = 0; sum2 = 0;
    for m = 1:M2:M
        m2 = min(M2, M - m + 1);
        logS = log(S0) * ones(1, m2);
        for n = 1:N
            BB = brownianBridge(1, h);  % Generate a Brownian bridge for the
                interval
            dW = BB(end);  % Use the endpoint of the bridge as the increment
            logS = logS + (r - 0.5 * sigma^2) * h + sigma * dW;
        end
        S = exp(logS);
        P = exp(-r * T) * max(S - K, 0);

        sum1 = sum1 + sum(P);
        sum2 = sum2 + sum(P.^2);
    end
    hh(p) = h;
```

```
    err1(p) = sum1 / M - Ve;
    err2(p) = 3 * sqrt((sum2 / M - (sum1 / M)^2) / (M - 1));
end

% Plotting results for weak convergence
figure; loglog(hh, abs(err1), 'b-*', hh, err2, 'r-*');
title('Weak convergence with Brownian Bridge');
xlabel('h'); ylabel('Error');
legend('Absolute Weak error', '3*STD MC error', 'location', 'NorthWest');
```

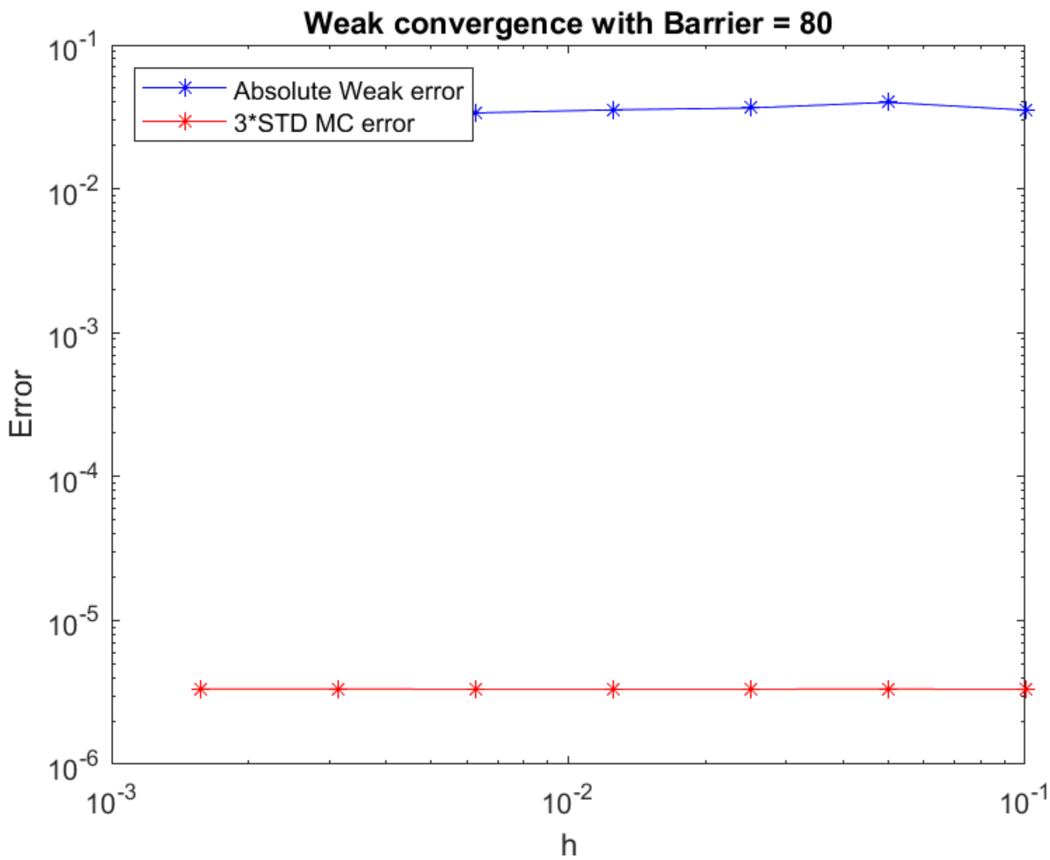

Figure 12: Weak Convergence with Brownian Bridge

Incorporating a Brownian bridge at each timestep improves the accuracy of simulating the barrier crossings.HAving a smaller error, that can be compared to previous iteration taking into account the scale they have been plotted in.

vi) Solve numerically without using Euler-Maruyama nor timestep h, and study weak convergence.

For this section, it's important to understand two things:

- The closed-form solution of the probability of the GBM is known, so no simulation of the trajectory is needed. Instead, we need to simulate the probabilities of the values using the known probability distribution.
- A brownian bridge can be implemented, fixing the time at $t = 0$ and $t = T$.

This is the code we created to solve this task:

```
% Parameters
```

```
S0 = 100;
K = 100;
sigma = 0.3;
r = 0.1;
T = 0.2;
B = 85;
exact_solution = 6.3076; % Given exact solution

% Number of simulations
num_simulations = 10000;

% Initialize the payoff array
payoff = zeros(num_simulations, 1);

% Function to generate Brownian Bridge
brownian_bridge = @(t1, t2, x1, x2, n) linspace(x1, x2, n) + sqrt(t2-t1)*randn(1,n);

% Perform simulations
for i = 1:num_simulations
    % Generate the Brownian Bridge from t=0 to t=T
    n = 100; % Number of time steps
    dt = T / n;
    t = 0:dt:T;
    W = cumsum([0, sqrt(dt)*randn(1, n)]);
    S = S0 * exp((r - 0.5 * sigma^2) * t + sigma * W);

    % Check if the barrier is breached at any time step
    if any(S <= B)
        payoff(i) = 0; % Barrier breached, payoff is zero
    else
        payoff(i) = exp(-r * T) * max(S(end) - K, 0); % Barrier not breached
    end
end

% Calculate the estimated price
estimated_price = mean(payoff);
standard_error = std(payoff) / sqrt(num_simulations);

% Display the results
fprintf('Estimated Price: %.4f\n', estimated_price);
fprintf('Standard Error: %.4f\n', standard_error);
fprintf('Exact Solution: %.4f\n', exact_solution);
fprintf('Error: %.4f\n', abs(estimated_price - exact_solution));
```

After running the code above, we obtain the following results.

- Estimated Price: 6.4501
- Standard Error: 0.0928
- Exact Solution: 6.3076
- Error: 0.1425

As we, see the error is not considerably large but other methods might be used to obtain the solution yielding smaller errors.

vii) Which is the most efficient algorithm? (i.e. the one which takes least time to attain a given error). Make your argument using an error vs. computational time plot. Efficiency in algorithmic trading or financial simulations is often a balance between computational speed and accuracy.

Plotting error against computational time provides a clear, visual representation of which algorithm offers the best trade-off between speed and precision.

CODE 12: CPU time vs Abs error

```
function main3()
    % Define parameters for GBM
    S0 = 100;  % Initial stock price
```

```
    K = 100;   % Strike price
    sigma = 0.3; % Volatility
    r = 0.1;    % Risk-free rate
    T = 0.2;    % Time to maturity
    B = 85;     % Barrier level
    Ve = 6.3076; % Known exact solution for comparison

    methods = {@method1, @method2, @method3, @method4};
    methodNames = {'Euler-Maruyama', 'Log Transformation', 'Barrier Option', 'Brownian Bridge'};
    numMethods = length(methods);

    cpuTimes = zeros(numMethods, 1);
    errors = zeros(numMethods, 1);

    for i = 1:numMethods
        tic;
        estimatedValue = methods{i}(S0, K, r, T, sigma, B, Ve);
        cpuTimes(i) = toc;

        errors(i) = abs(estimatedValue - Ve);
    end

    figure;
    scatter(cpuTimes, errors, 100, 'filled');
    text(cpuTimes, errors, methodNames, 'VerticalAlignment', 'top', 'HorizontalAlignment', 'right');
    title('CPU Time vs Accuracy');
    xlabel('CPU Time (seconds)');
    ylabel('Absolute Error');
    grid on;
end

function value = method1(S0, K, r, T, sigma, B, Ve)
    % Euler-Maruyama
    dt = T / 100;
    N = T / dt;
    S = S0;
    for i = 1:N
        dW = sqrt(dt) * randn;
        S = S * (1 + r * dt + sigma * dW);
    end
    value = exp(-r * T) * max(S - K, 0);
end

function value = method2(S0, K, r, T, sigma, B, Ve)
    dt = T / 100;
    N = T / dt;
    logS = log(S0);
    for i = 1:N
        dW = sqrt(dt) * randn;
        logS = logS + (r - 0.5 * sigma^2) * dt + sigma * dW;
    end
    S = exp(logS);
    value = exp(-r * T) * max(S - K, 0);
end

function value = method3(S0, K, r, T, sigma, B, Ve)
    dt = T / 100;
    N = T / dt;
    S = S0;
    knockedOut = false;
    for i = 1:N
        dW = sqrt(dt) * randn;
        S = S * (1 + r * dt + sigma * dW);
        if S <= B
            knockedOut = true;
            break;
        end
    end
    if knockedOut
```

```
        value = 0;
    else
        value = exp(-r * T) * max(S - K, 0);
    end
end

function value = method4(S0, K, r, T, sigma, B, Ve)
    dt = T / 100;
    N = T / dt;
    S = S0;
    for i = 1:N
        if i == 1
            nextW = sqrt(T) * randn;
        end
        currentW = sqrt(dt) * i / N * nextW;
        S = S * (1 + r * dt + sigma * (currentW - (i-1)/N * nextW));
        nextW = currentW;
    end
    value = exp(-r * T) * max(S - K, 0);
end
```

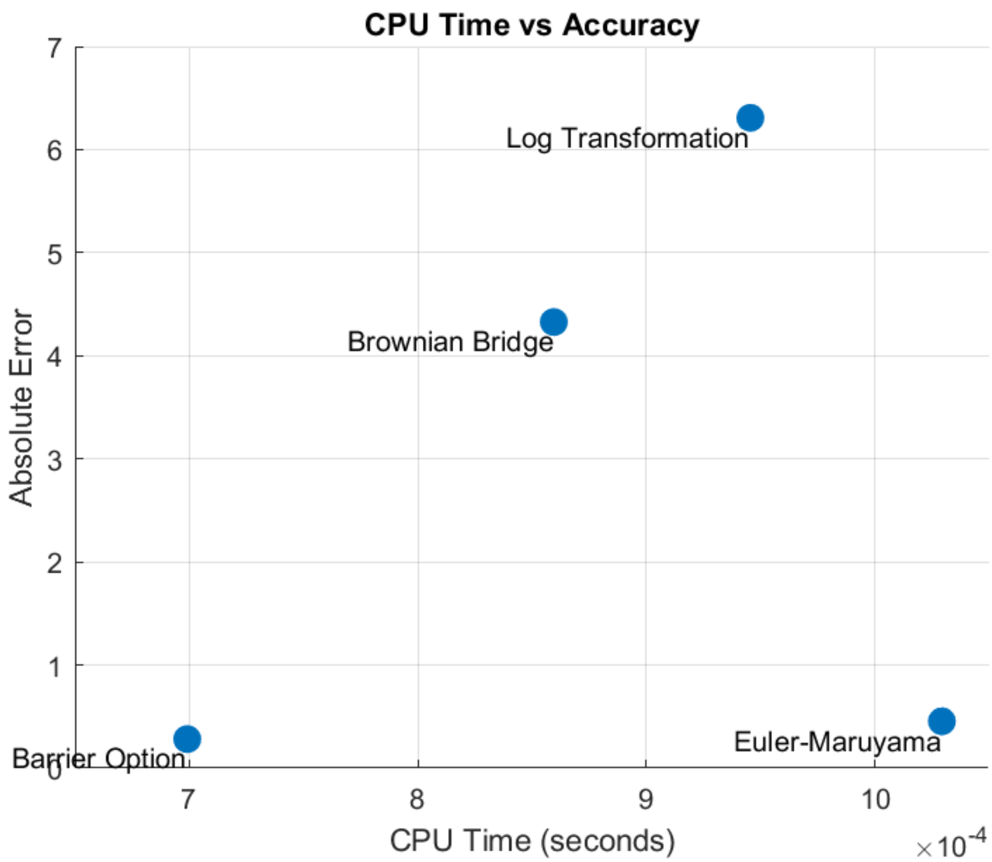

Figure 13: CPU time vs Abs error

The results we observed were quite surprising, particularly in terms of error rates. It's understandable that the Euler-Maruyama method is the most time-consuming due to its computational intensity, followed by the log transformation. Interestingly, more advanced techniques like barriers and Brownian Bridge required less CPU time. However, it was unexpected to find that the logarithmic transformation resulted in greater errors compared to the Euler-Maruyama method.

viii) Can you get even better speed with variance reduction?

The use of variance reduction techniques such as antithetic variates, control variates is discussed as a method to enhance the accuracy and efficiency of stochastic simulations. By reducing the variance of the estimator, these techniques enable more accurate results with fewer simulation runs, which is especially beneficial in high-frequency trading or risk management scenarios where speed and accuracy are critical. So we think a better speed will be possible implementing these techniques.

Exercise 4. According to the Malthusian population model, the number of individuals grows at a constant rate $dN/N = g\,dt$, with $g > 0$ and $N_0 = N(t = 0)$. The stochastic version (allowing for noise in g) is the SDE $dN_t = gN_t\,dt + \nu N_t\,dW_t$, with $\nu > 0$. The population is declared extinct if $N_T < 1$—otherwise it survives after time T. Pick $g = .03$, $\nu = 0.5$, $N_0 = 3$, $T = 10$.

i) The probability of population survival after time T can be written as $E[\phi(T)]$. Write down a formula for $\phi(T)$.

Let

$$\phi(T) = \begin{cases} 1 & \text{if } N_T \geq 1, \\ 0 & \text{if } N_T < 1. \end{cases}$$

Given the SDE:

$$dN_t = gN_t\,dt + \nu N_t\,dW_t,$$

the solution is:

$$N_T = N_0 \exp\left(\left(g - \frac{\nu^2}{2}\right)T + \nu W_T\right),$$

Nt is the population size at time t, g is the deterministic growth rate of the population, v is the noise in the population growth.

The probability of survival is:

$$P(N_T \geq 1) = P\left(N_0 \exp\left(\left(g - \frac{\nu^2}{2}\right)T + \nu W_T\right) \geq 1\right),$$

$$P\left(\left(g - \frac{\nu^2}{2}\right)T + \nu W_T \geq \ln\left(\frac{1}{N_0}\right)\right),$$

$$P\left(W_T \geq \frac{1}{\nu}\left(\ln\left(\frac{1}{N_0}\right) - \left(g - \frac{\nu^2}{2}\right)T\right)\right),$$

$$P\left(\frac{W_T}{\sqrt{T}} \geq \frac{1}{\nu\sqrt{T}}\left(\ln\left(\frac{1}{N_0}\right) - \left(g - \frac{\nu^2}{2}\right)T\right)\right),$$

Let $Z \sim \mathcal{N}(0, 1)$:

$$P\left(Z \geq \frac{1}{\nu\sqrt{T}}\left(\ln\left(\frac{1}{N_0}\right) - \left(g - \frac{\nu^2}{2}\right)T\right)\right),$$

Using the CDF of the standard normal distribution Φ:

$$P(Z \geq x) = 1 - \Phi(x),$$

we get:

$$P(N_T \geq 1) = 1 - \Phi\left(\frac{1}{\nu\sqrt{T}}\left(\ln\left(\frac{1}{N_0}\right) - \left(g - \frac{\nu^2}{2}\right)T\right)\right).$$

Thus, the formula for $\phi(T)$ is:

$$\phi(T) = 1 - \Phi\left(\frac{1}{\nu\sqrt{T}}\left(\ln\left(\frac{1}{N_0}\right) - \left(g - \frac{\nu^2}{2}\right)T\right)\right).$$

ii) Let's find a *semianalytical solution*. Observe that $\phi(T)$ resembles the payoff of a barrier option. Combine d/o Calls with different signs and strikes to make this exact. Then, use the findings of Exercise 1 to compute the sought-for survival probability using the solution of vanilla Calls (using `blsprice`). Check against a numerical solution for the former.

Considering it as a down-and-out barrier option. We will transform the problem into the form of a down-and-out call option.
Transformation:
Population N_t is analogous to stock price S_t. The drift g and volatility ν describe the dynamics:

$$dS_t = gS_t dt + \nu S_t dW_t.$$

To compute the survival probability using the solution of vanilla calls (using `blsprice`), we use the formula:

$$C_{d.o.}(t, S) = C_v(t, S) - \left(\frac{S}{B}\right)^{1-\frac{2r}{\sigma^2}} C_v(t, BS),$$

where:

- $C_{d.o.}(t, S)$ is the down-and-out call option price.
- $C_v(t, S)$ is the price of a vanilla call option.
- S is the initial stock price (initial population).
- B is the barrier level.
- r is the risk-free interest rate.
- σ is the volatility of the stock (population).

CODE 13: Numerical vs Analitical Comparison

```
% Parameters
g = 0.03;
nu = 0.5;
N0 = 3;
T = 10;
numSimulations = 10000;
dt = 0.01; % time step size
numSteps = T / dt;

% Pre-allocate array to store the final population values
finalPopulations = zeros(numSimulations, 1);

for i = 1:numSimulations
    % Initialize the population
    Nt = N0;
    % Simulate the SDE over time
    for t = 1:numSteps
        dWt = sqrt(dt) * randn; % Brownian motion increment
        Nt = Nt + g * Nt * dt + nu * Nt * dWt; % Euler-Maruyama method
    end
    finalPopulations(i) = Nt;
end

% Calculate the proportion of simulations where the population is >= 1 at T = 10
numericalSurvivalProbability = mean(finalPopulations >= 1);

% Analytical survival probability
ln_term = log(1 / N0);
```

```
analyticalSurvivalProbability = 1 - normcdf((ln_term - (g - (nu^2 / 2)) * T) / (nu * sqrt(T)));

% Display the results
disp(['Numerical estimated probability of survival: ', num2str(numericalSurvivalProbability)]);
disp(['Analytical probability of survival: ', num2str(analyticalSurvivalProbability)]);
```

Numerical estimated probability of survival: 0.5301
Analytical probability of survival: 0.53744

iii) Write a Matlab code which estimates the probability that the population exceeds $2N_0$ at any time before T. Use the shifting method for accuracy. *Hint:* you must make sure it has not become extinct before! How many barriers you have now? How do you apply the shifting?

To estimate the probability that the population exceeds $2N_0$ at any time before T, we can write a MATLAB code. We need to ensure that the population does not become extinct before.

For this problem, we employ two barriers: one to prevent extinction and the other to avoid overestimating population growth 2No. The lower (down) barrier is shifted upwards by an epsilon to more accurately capture potential extinction events, ensuring that even small population decreases are noted before the model reports extinction. On the other hand, the upper (up) barrier is adjusted downwards by an epsilon, refining the model to prevent the overstatement of growth. This careful adjustment of both barriers helps mitigate errors due to the discretization of the simulation process.

CODE 14: Malthusian

```
function [survivalProb, stdError, excessRisk, computationTime, variance, totalPaths] = malthusian(populationSize, timeSteps, seed, varargin)
    if seed ~= -1
        rng(seed, 'twister');
    end
    displayDetails = false;
    if nargin > 4
        displayDetails = true;
    end
    initialPopulation = 3;
    totalDuration = 10;
    volatility = 0.5;
    growthRate = 0.03;
    timeStepSize = totalDuration / timeSteps;
    batchSize = min(populationSize, 1e5);
    pathsProcessed = 0;
    totalHits = 0;
    sumOfSquares = 0;
    barrierHits = 0;

    downOutBarrier = exp(log(initialPopulation) + 0.5826 * volatility * sqrt(timeStepSize));
    upInBarrier = exp(log(2 * initialPopulation) - 0.5826 * volatility * sqrt(timeStepSize));

    tic
    while pathsProcessed < populationSize
        populationLog = NaN(batchSize, timeSteps + 1);
        populationLog(:, 1) = log(initialPopulation);
        hitFlag = NaN(batchSize, 1);

        for step = 1:timeSteps

            populationLog(:, step + 1) = populationLog(:, step) + (growthRate - volatility^2 / 2) * timeStepSize + volatility * sqrt(timeStepSize) * randn(batchSize, 1);

            for i = 1:batchSize
                if isnan(hitFlag(i))
```

```
                    if populationLog(i, step + 1) <= log(downOutBarrier) % Down-and-out
                        hitFlag(i) = 0;
                    elseif populationLog(i, step + 1) >= log(upInBarrier) % Up-and-in
                        hitFlag(i) = 1;
                    end
                end
            end
        end

        hitFlag(isnan(hitFlag)) = 0;
        totalHits = totalHits + sum(hitFlag);
        sumOfSquares = sumOfSquares + sum(hitFlag .* 2);
        barrierHits = barrierHits + sum(hitFlag);
        pathsProcessed = pathsProcessed + batchSize;
    end

    survivalProb = totalHits / pathsProcessed;
    variance = sumOfSquares / pathsProcessed - survivalProb^2;
    stdError = 3 * sqrt(variance / pathsProcessed);
    computationTime = toc;

    if displayDetails
        fprintf('N (effective)=%d, time step size=%g: Probability=%g +/- %g, Computation Time=%g, %% hit barrier=%g \n', ...
            pathsProcessed, timeStepSize, survivalProb, stdError, computationTime, 100 * (barrierHits / pathsProcessed));
    end
end
```

This code estimates the probability that the population exceeds $2N_0$ with a probability of 6.02% at any time before T using a stochastic simulation. It checks for extinction at each step and stops simulation for extinct paths.