

Numerical Finance Reading Course

Sheet 7 - Sample Solution

Exercise 1: Black Scholes PDE

Let $S(t)$ denote some asset following a geometric Brownian motion:

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t).$$

Let $B(t)$ denote a risk-free bond:

$$dB(t) = rB(t)dt,$$

and let $Y(t)$ denote a (normalized) replicating portfolio

$$Y(t) = c_1(t)B(t) + c_2(t)S(t) - V(S(t), t) \quad \text{for some } c_1(t), c_2(t).$$

Assume that $Y(t)$ is self-financing, i.e. it holds

$$dY(t) = c_1(t)dB(t) + c_2(t)dS(t) - dV(S(t), t).$$

Derive the Black Scholes PDE in the following way:

- Calculate $dV(S(t), t)$ with the help of the Ito formula:

$$\begin{aligned} dX_t &= a(t)dt + b(t)dW(t) \\ \implies df(X(t)) &= \left(f_t(X(t)) + a(t)f_X(X(t)) + \frac{1}{2}b^2(t)f_{XX}(X(t)) \right) dt + b(t)f_X(X(t))dW(t). \end{aligned}$$

- Calculate $dY(t)$ as above.
- Use the fact that $Y(t)$ is riskless to deduce that $dY(t) = Y(t)r dt$ and compare the two representations of $dY(t)$ to obtain the PDE.

Solution:

Using the Ito-Lemma, it holds

$$dV(S(t), t) = \left(V_t(S(t), t) + \mu S_t V_S(S(t), t) + \frac{1}{2}\sigma^2 S^2(t) V_{SS}(S(t), t) \right) dt + \sigma S(t) V_S(S(t), t) dW(t).$$

Inserting this and $dB(t)$, $dS(t)$ into $dY(t)$ (and dropping the arguments for shorthand notation), we obtain

$$\begin{aligned}
dY(t) &= c_1(t)B(t)r dt + c_2(t)S(t)(\mu dt + \sigma dW(t)) - \left(V_t + \mu S_t V_S + \frac{1}{2} \sigma^2 S^2(t) V_{SS} \right) dt \\
&\quad - \sigma S(t) V_S dW(t) \\
&= \underbrace{\left[c_1(t)B(t)r + c_2(t)S(t)\mu - \left(V_t + \mu S_t V_S + \frac{1}{2} \sigma^2 S^2(t) V_{SS} \right) \right]}_{(*)} dt \\
&\quad + \underbrace{[c_2(t)S(t)\sigma - S(t)\sigma V_S]}_{(**)} dW(t)
\end{aligned}$$

Since $Y(t)$ is a replicating portfolio, it is riskless and hence $dY(t) = Y(t)r dt$. Thus

$$\begin{aligned}
(*) &= rY(t) \\
(**) &= 0.
\end{aligned}$$

This yields

$$\begin{aligned}
0 &= c_2(t)S(t)\sigma - S(t)\sigma V_S \quad \implies \quad c_2(t) = V_S(S(t), t), \\
rY(t) &= r(c_1(t)B(t) + c_2(t)S(t) - V(S(t), t)) \\
&\stackrel{!}{=} c_1(t)B(t)r + c_2(t)S(t)\mu - \left(V_t + \mu S_t V_S + \frac{1}{2} \sigma^2 S^2(t) V_{SS} \right) \\
&= c_1(t)rB(t) - V_t - \frac{1}{2} \sigma^2 S^2(t) V_{SS}. \\
\implies \quad rS(t)V_S - rV &= -V_t - \frac{1}{2} \sigma^2 S^2(t) V_{SS}. \\
\implies \quad V_t + \frac{1}{2} \sigma^2 S^2(t) V_{SS} + rS(t)V_S - rV &= 0.
\end{aligned}$$

Exercise 2: Simulating Wiener Processes

There are several methods to generate paths of a Wiener process. We will consider here the two most common ones.

a) Random Walk:

Making use of the fact that the increments of a Wiener process W are independently distributed with $W_t - W_s \sim N(0, t - s)$ for all $0 \leq s < t \leq T$, we can simulate paths in the following way. Let $0 = t_0 < t_1 < \dots < t_N = T$, $Z_1, \dots, Z_N \sim N(0, 1)$. Then define

$$\begin{aligned}
W_0 &= 0, \\
W_{t_{i+1}} &= W_{t_i} + \sqrt{t_{i+1} - t_i} Z_{i+1} \quad \text{for } i = 1, \dots, N.
\end{aligned}$$

(Note that this corresponds to the Euler-Maruyama method for the PDE $dX_t = dW_t$, $X_0 = 0$.)

b) **Brownian Bridge:**

One can also start by generating the endpoint W_T of the process, and then fill in the other values using the distribution of W_t conditional on the already generated points. This procedure is known as Brownian Bridge. More precisely, consider $N = 2^n$ and $t_i - t_{i-1} = \frac{T}{N}$ for all $i = 1, \dots, N$. One then generates first W_T , then $W_{\frac{T}{2}}$ based on W_0 and W_T , then $W_{\frac{T}{4}}, W_{\frac{3T}{4}}$ based on $W_0, W_{\frac{T}{2}}$ and W_T , etc.

Suppose one has already calculated the $(j-1)$ th refinement $W_0, W_{\frac{T}{2^{j-1}}}, W_{\frac{2T}{2^{j-1}}}, \dots, W_{\frac{2^{j-1}T}{2^{j-1}}}$, $j < n - 1$. Due to the independence of the increments, the value of W_s depends only on the two nearest values $W_{s_k} := W_{\frac{kT}{2^{j-1}}} < s < W_{\frac{(k+1)T}{2^{j-1}}} =: W_{s_{k+1}}$. In our construction, we always consider $s = \frac{1}{2} \left(\frac{kT}{2^{j-1}} + \frac{(k+1)T}{2^{j-1}} \right)$, i.e. the midpoint between two already calculated time points. In that case, the conditional distribution of W_s is given by

$$(W_s \mid W_{s_k} = w_{s_k}, W_{s_{k+1}} = w_{s_{k+1}}) \sim N \left(\frac{w_{s_k} + w_{s_{k+1}}}{2}, \frac{T}{2^{j+1}} \right).$$

This yields the following construction, using $Z_1, \dots, Z_N \sim N(0, 1)$:

$$\begin{aligned} W^0(0) &= 0, W^0(T) = \sqrt{T}Z_1, \\ W^1(t) &= \begin{cases} \frac{1}{2}(W^0(0) + W^0(T)) + \sqrt{\frac{T}{2^2}}Z_2, & t = \frac{1}{2}, \\ W^0(t), & t = 0, T, \end{cases} \\ W^j(t) &= \begin{cases} \frac{1}{2} \left(W^{j-1} \left(\frac{2kT}{2^j} \right) + W^{j-1} \left(\frac{(2k+2)T}{2^j} \right) \right) + \sqrt{\frac{T}{2^{j+1}}}Z, & t = \frac{(2k+1)T}{2^j}, \\ & k = 0, \dots, 2^{j-1} - 1 \\ W^{j-1} \left(\frac{kT}{2^{j-1}} \right), & t = \frac{2kT}{2^j}, \\ & k = 0, \dots, 2^{j-1} \end{cases} \end{aligned}$$

for $j = 2, \dots, n$. (Note that the second line in $W^j(t)$ corresponds just to the values of W that have already been calculated).

Generate and plot paths of $\{W(t), 0 \leq t \leq 1\}$ using both methods for different N (for example $N = 4, 8, 32, 512, 1024$).

Solution:

a) **Random Walk:**

```
#include <iostream>
#include <cmath>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

/*
 * Linear Congruential Generator
 */
class LCG {
```

```

private:
    long long int a;
    long long int b;
    long long int M;

    long long int state;

public:
    LCG( long long int A, long long int B, long long int prime, int seed):
        a(A), b(B), M(prime), state((long long)seed) {};

    double uni(){
        state = (a* state + b) % M;
        return (double)state / (double)M;
    }
};

void BoxMuller(double &u1, double &u2){
    double theta = 2*M_PI*u2;
    double rho = sqrt(-2.*log(u1));

    u1 = rho*cos(theta);
    u2 = rho*sin(theta);
}

int main(int argc, char* argv[]){
    if(argc != 2){
        cout << "Usage: " << argv[0] << " n ntime steps" << endl;
        return 0;
    }

    /*
     * UNIX RAND Generator
     */
    long long int a = 1103515245LL;
    long long int b = 12345LL;
    long long int M = 2147483648LL;
    long long int seed = 1;
    LCG urand(a, b, M, seed);

    double Wt = 0.0;
    double T = 1;
    int n = atoi(argv[1]);
    int N = 1<<n;

    double deltaT = T/(double)N;
    double sqrtdeltaT = sqrt(T/(double)N);

    double u1, u2;

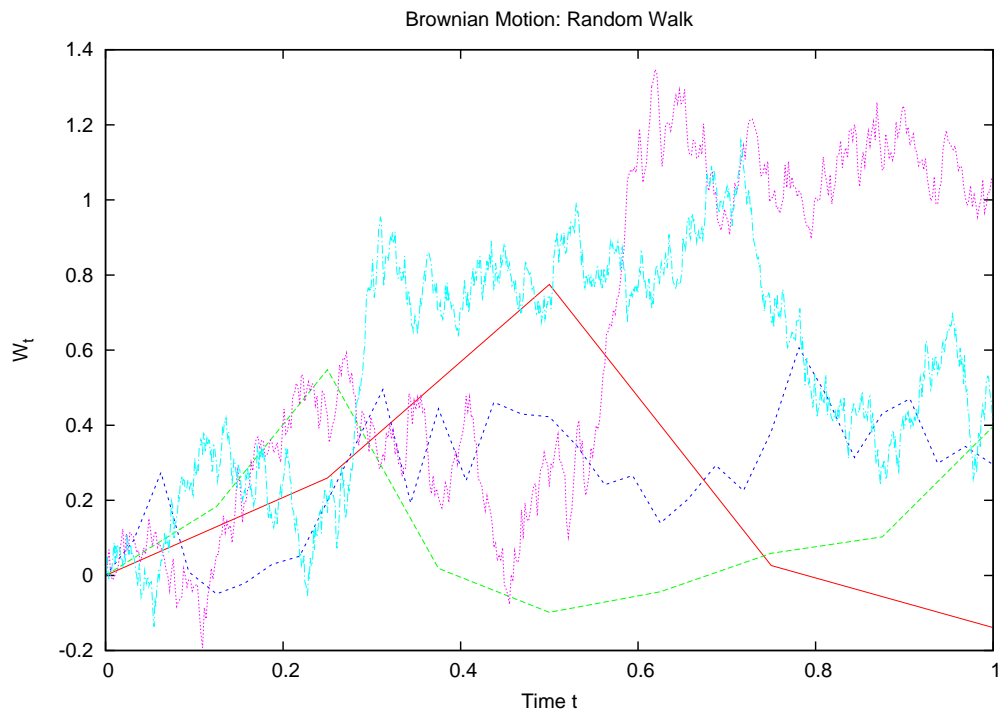
    cout << "0." << Wt << endl;

    for(int i = 1; i <= N; i += 2){
        u1 = urand.uni();
        u2 = urand.uni();
        BoxMuller(u1, u2);

        Wt += sqrtdeltaT*u1;
        cout << i*deltaT << "." << Wt << endl;
        Wt += sqrtdeltaT*u2;
        cout << (i+1)*deltaT << "." << Wt << endl;
    }

    return 0;
}

```



b) **Brownian Bridge:**

```

#include <iostream>
#include <cmath>
#include <vector>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

/*
 * Linear Congruential Generator
 */
class LCG {
private:
    long long int a;
    long long int b;
    long long int M;

    long long int state;

public:
    LCG( long long int A, long long int B, long long int prime, int seed):
        a(A), b(B), M(prime), state((long long)seed) {};

    double uni(){
        state = (a* state + b) % M;
        return (double)state / (double)M;
    }
};

void BoxMuller(double &u1, double &u2){
    double theta = 2*M_PI*u2;
    double rho = sqrt(-2.*log(u1));

    u1 = rho*cos(theta);
    u2 = rho*sin(theta);
}

```

```

int main(int argc, char* argv[]){

    if(argc != 2){
        cout << "Usage:_" << argv[0] << "_(2^n_time_steps)" << endl;
        return 0;
    }

    /*
     * UNIX RAND Generator
     */
    long long int a = 1103515245LL;
    long long int b = 12345LL;
    long long int M = 2147483648LL;
    long long int seed = 1;
    LCG urand(a, b, M, seed);

    double T = 1;
    int n = atoi(argv[1]);
    int N = 1<<n;
    double deltaT = 1./((double)N);
    vector<double> Wt(N+1);
    vector<double> X(N+1);

    // Fill X
    double u1, u2;
    for(int i = 0; i < N; i += 2){
        u1 = urand.uni();
        u2 = urand.uni();
        BoxMuller(u1, u2);

        X[i] = u1;
        X[i+1] = u2;
    }
    u1 = urand.uni();
    u2 = urand.uni();
    BoxMuller(u1, u2);
    X[N] = u1;

    // Construct Brownian Bridge
    int count = 0;
    Wt[0] = 0.;
    Wt[N] = T*X[N];
    for(int i = 1; i <= n; i++){
        for(int k = 0; k < 1<<(i-1); k++){
            Wt[(2*k+1)*(1<<(n-i))] = 0.5*(Wt[2*k*(1<<(n-i))] + Wt[(2*k+2)*(1<<(n-i))])
            + X[count]/sqrt(1<<(i+1));
            count++;
        }
    }

    // Print
    for(int i = 0; i <= N; i++){
        cout << i*deltaT << "_" << Wt[i] << endl;
    }

    return 0;
}

```

