

Numerical Finance Reading Course

Sheet 8 - Sample Solution

Exercise 1: Strong Consistency of Euler-Maruyama (Prop. 5.6.9)

Show that the Euler-Maruyama scheme is strongly consistent with $c(\delta) = 0$ under the assumptions of Theorem 5.2.1.

Hints:

- ΔW_n is independent of \mathcal{A}_{τ_n} .
- τ_n, τ_{n+1} are \mathcal{A}_{τ_n} -measurable, i.e. $\mathbb{E}[\tau_n | \mathcal{A}_{\tau_n}] = \tau_n$.

Solution:

Consider a fixed $Y_n^\delta = y$. Then

$$\begin{aligned} & \mathbb{E} \left[\left| \mathbb{E} \left[\frac{Y_{n+1}^\delta - Y_n^\delta}{\Delta_n} \middle| \mathcal{A}_{\tau_n} \right] - a(\tau_n, Y_n^\delta) \right|^2 \right] \\ &= \mathbb{E} \left[\left| \mathbb{E} \left[\frac{y + a(\tau_n, y)\Delta_n + b(\tau, y)\Delta W_n - y}{\Delta_n} \middle| \mathcal{A}_{\tau_n} \right] - a(\tau_n, Y_n^\delta) \right|^2 \right] \\ &= \mathbb{E} \left[\left| \mathbb{E} \left[a(\tau_n, y) + \frac{b(\tau, y)\Delta W_n}{\Delta_n} \middle| \mathcal{A}_{\tau_n} \right] - a(\tau_n, Y_n^\delta) \right|^2 \right] \\ &= \mathbb{E} \left[\left| a(\tau_n, y) + \frac{b(\tau, y)}{\Delta_n} \underbrace{\mathbb{E} [\Delta W_n | \mathcal{A}_{\tau_n}]}_{=\mathbb{E}[\Delta W_n]=0} - a(\tau_n, Y_n^\delta) \right|^2 \right] \\ &= \mathbb{E} \left[\left| a(\tau_n, y) - a(\tau_n, Y_n^\delta) \right|^2 \right] \\ &= 0, \end{aligned}$$

since $a(\tau_n, y)$, $b(\tau_n, y)$ and Δ_n are \mathcal{A}_{τ_n} -measurable.

Moreover,

$$\begin{aligned}
& \mathbb{E} \left[\frac{1}{\Delta_n} \left| Y_{n+1}^\delta - Y_n^\delta - \mathbb{E}[Y_{n+1}^\delta - Y_n^\delta \mid \mathcal{A}_{\tau_n}] - b(\tau_n, Y_n^\delta) \Delta W_n \right|^2 \right] \\
&= \mathbb{E} \left[\frac{1}{\Delta_n} \left| y + a(\tau_n, y) \Delta_n + b(\tau, y) \Delta W_n - y - \mathbb{E}[y + a(\tau_n, y) \Delta_n + b(\tau, y) \Delta W_n - y \mid \mathcal{A}_{\tau_n}] \right. \right. \\
&\quad \left. \left. - b(\tau_n, y) \Delta W_n \right|^2 \right] \\
&= \mathbb{E} \left[\frac{1}{\Delta_n} \left| a(\tau_n, y) \Delta_n + b(\tau, y) \Delta W_n - a(\tau_n, y) \Delta_n + b(\tau, y) \underbrace{\mathbb{E}[\Delta W_n \mid \mathcal{A}_{\tau_n}]}_{=0} - b(\tau_n, y) \Delta W_n \right|^2 \right] \\
&= 0.
\end{aligned}$$

Exercise 2: Option Pricing with Euler-Maruyama

European Barrier Options are options that depend on the underlying S to reach a given barrier H . For example, a European Up&Out Put becomes worthless if $S_t > H$ for any $t \in [0, T]$, so that

$$V_T = (K - S_T)^+ \mathbb{1}_{\{S_t < H \ \forall t \in [0, T]\}}.$$

The analytic solution for such a put is given by

$$\begin{aligned}
P_0^{Up\&Out} &= e^{-rT} \mathbb{E}[V_T] \\
&= K e^{-rT} \left(\Phi(-d + \sigma\sqrt{T}) - \left(\frac{H}{S}\right)^{2\lambda-2} \Phi(-y + \sigma\sqrt{T}) \right) \\
&\quad - S_0 \left(\Phi(-d) - \left(\frac{H}{S}\right)^{2\lambda} \Phi(-y) \right),
\end{aligned}$$

where

$$\lambda = \frac{r + 0.5\sigma^2}{\sigma^2}, \quad y = \frac{\ln\left(\frac{H^2}{SK}\right)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}, \quad d = \frac{\ln\left(\frac{S}{K}\right) + (r + 0.5\sigma^2)T}{\sigma\sqrt{T}}.$$

To estimate $P_0^{Up\&Out}$ numerically, one generates M replications of the path of S_t (using for example Euler-Maruyama), calculates the payoff $V_T^{(1)}, \dots, V_T^{(M)}$ for each path and sets

$$\hat{P}_0^{Up\&Out} = e^{-rT} \frac{1}{M} \sum_{i=1}^M V_T^{(i)}.$$

Assume that S is geometric Brownian motion, i.e.

$$dS_t = rS_t dt + \sigma S_t dW_t,$$

and let $S_0 = 10$, $K = 12$, $H = 13$, $r = 0.04$, $\sigma = 0.4$ and $T = 2$. Calculate the price of a European Up&Out Put using for the simulation of the Brownian motion W_t

- a) a random walk construction.
- b) a Brownian Bridge construction.
- c) a Brownian Bridge construction with Quasi-Monte Carlo numbers (drawing one QMC number for each path). Note that this construction is favorable for QMC, as the first dimensions of each point are those that determine the shape of the path. You can find instructions on how to generate Sobol numbers using the libraries *QuantLib* and *Boost* on the course homepage.

Compare the obtained prices for different M (e.g. $M = 2^2, \dots, 2^{13}$) and plot the convergence rates. You can use for example $N = 2^{13}$ time steps.

Solution:

a) **Random Walk:**

```

#include <iostream>
#include <cmath>
#include <vector>
#include <boost/random.hpp>
#include <time.h>

using namespace std;
using namespace boost;

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

    if(argc != 3){
        cerr << "Usage: ./ " << argv[0] << " ./N.(2^N.time_steps) ./M.(2^M.paths)" << endl;
        return 0;
    }

    double T = 2;
    double S0 = 10;
    double K = 12;
    double H = 13;
    double r = 0.04;
    double sigma = 0.4;
    double sol = 2.047849;
    int Nexpt = atoi(argv[1]);
    int N = 1 << Nexpt;

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

    vector<double> Y(N+1);

    double VT;
    double V0;
    int Mexpt = atoi(argv[2]);

    mt19937 generator;
    generator.seed(time(0));
    normal_distribution<> norm_dist;
    variate_generator<mt19937&,normal_distribution<> > norm_rnd(generator, norm_dist);

    // Convergence: for different M...
    for(int j = 1; j <= Mexpt; j++){
        V0 = 0;

        // .. generate M paths
        for(int n = 1; n <= 1<<j; n++){
            Y[0] = S0;
            // Euler-Maruyama scheme
            for(int i = 1; i <= N; i++){

```

```

        Y[i] = Y[i-1] + r*Y[i-1]*deltaT + sigma*Y[i-1]*sqrtdeltaT*norm_rnd();
    }

    // Calculation of V_T
    bool out = false;
    for(int i=1;i<=N;i++){
        if(Y[i] >= H){
            out = true;
            break;
        }
    }
    if(out == false){
        VT = (K - Y[N]) > 0 ? K-Y[N] : 0;
    }
    else{
        VT = 0;
    }

    V0 += VT;

}

// Print M, V_0 - sol, V_0
cout << (1<<j) << " " << abs( (V0 / ((1<<j) * exp(r*T))) - sol)
    << " " << (V0 / ((1<<j) * exp(r*T))) << endl;
}

return 0;
}

```

b) Brownian Bridge:

```

#include <iostream>
#include <cmath>
#include <vector>
#include <boost/random.hpp>
#include <time.h>

using namespace std;
using namespace boost;

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

    if(argc != 3){
        cerr << "Usage: " << argv[0] << " _N_(2^N_time_steps) _M_(2^M_paths)" << endl;
        return 0;
    }

    double T = 2;
    double S0 = 10;
    double K = 12;
    double H = 13;
    double r = 0.04;
    double sigma = 0.4;
    double sol = 2.047849;
    int Nexpt = atoi(argv[1]);
    int N = 1<<Nexpt;

    double deltaT = T/(double)N;

    vector<double> Wt(N+1);
    vector<double> Y(N+1);

    double VT;
    double V0;
    int Mexpt = atoi(argv[2]);

    mt19937 generator;
    generator.seed(time(0));
    normal_distribution<> norm_dist;

```

```

variate_generator<mt19937&,normal_distribution<> > norm_rnd(generator , norm_dist);

for(int j = 1; j <= Mexp; j++){
    V0 = 0;

    for(int n = 1; n <= 1<<j; n++){

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

        // Euler-Maruyama
        Y[0] = S0;
        for(int i = 1; i <= N; i++){
            Y[i] = Y[i-1] + r*Y[i-1]*deltaT + sigma*Y[i-1]*(Wt[i] - Wt[i-1]);
        }

        // Option Value at t=T
        bool out = false;
        for(int i=1;i<=N;i++){
            if(Y[i] >= H){
                out = true;
                break;
            }
        }
        if(out == false){
            VT = (K - Y[N]) > 0 ? K-Y[N] : 0;
        }
        else{
            VT = 0;
        }
        V0 += VT;
    }

    // Print M, V_0 - sol, V_0
    cout << (1<<j) << " " << abs( (V0 / ((1<<j) * exp(r*T))) - sol) << " "
        << V0 / ((1<<j) * exp(r*T)) << endl;
}

return 0;
}

```

c) Brownian Bridge with QMC:

```

#include <iostream>
#include <cmath>
#include <vector>
#include <ql/math/randomnumbers/sobolrsg.hpp>

using namespace std;
using namespace QuantLib;

typedef std::vector<QuantLib::Real> QMCPPoint;

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 != 3){

```

```

    cerr << "Usage: " << argv[0] << " -N_(2^N_time_steps) -M_(2^M_paths)" << endl;
    return 0;
}

// Definition of Problem
double T = 2;
double S0 = 10;
double K = 12;
double H = 13;
double r = 0.04;
double sigma = 0.4;
double sol = 2.047849;
int Nexpt = atoi(argv[1]);
int N = 1<<Nexpt;
    double deltaT = T/(double)N;

// Variables for stochastic process
vector<double> Wt(N+1);
vector<double> Y(N+1);
SobolRsg sobol(N, 1);
QMCPPoint qmcp;
int qmcindex;

// Simulation variables
double VT;
double V0;
int Mexpt = atoi(argv[2]);

// Simulation runs for different M
for(int j = 1; j <= Mexpt; j++){
    V0 = 0;
    sobol.skipTo(4096);

    // Simulate 2^j paths
    for(int n = 1; n <= 1<<j; n++){

        // Generation of (N+1)-dimensional QMG-Number
        qmcp = sobol.nextSequence().value;
        for(int i = 0; i < N; i += 2){
            BoxMuller(qmcp[i], qmcp[i+1]);
        }

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

        // Euler-Maruyama
        Y[0] = S0;
        for(int i = 1; i <= N; i++){
            Y[i] = Y[i-1] + r*Y[i-1]*deltaT + sigma*Y[i-1]*(Wt[i] - Wt[i-1]);
        }

        // Option Value at t=T
        bool out = false;
        for(int i=1;i<=N;i++){
            if(Y[i] >= H){
                out = true;
                break;
            }
        }
        if(out == false){
            VT = (K - Y[N]) > 0 ? K-Y[N] : 0;
        }
    }
}

```

```

    }
    else{
        VT = 0;
    }
    V0 += VT;
}
cout << (1<<j) << " " << abs( (V0 / ((1<<j) * exp(r*T))) - sol) << " "
    << V0 / ((1<<j) * exp(r*T)) << endl;
}
return 0;
}

```



