Dr. Roxana Dumitrescu
Department of Mathematics
King's College London
email: roxana.dumitrescu@kcl.ac.uk

Exercises C++
Sheet 7

1 Binomial pricer (continuation)

We have written several functions in order to compute an approximation of the price of American options and European options (calls and puts) in the BlackSholes model by means of the binomial approximation. What aspects of the design did you find unsatisfactory? What solution do you propose?

2 Monte-Carlo pricer

2.1 Simulation

Problem 1. Write a function **randuniform(int n)** which returns a vector of uniformly distributed random numbers in the range $(0, 1)$.

Problem 2. Write a function **randn(int n)** which returns a vector of normally distributed random numbers with mean 0 and standard deviation 1 ($\mathcal{N}(0, 1)$), using the function norminv (implemented in the first practical) and the fact that norminv(u) $\sim \mathcal{N}(0, 1)$, with u an uniform random variable in $(0, 1)$.

Problem 3 [Box-Muller algorithm]. An alternative way to generate normally distributed random numbers is to use the Box-Muller algorithm. To do this, one first generates two uniformly distributed random variables u_1 and u_2 in the interval $(0, 1)$. Define:

$$Z_1 = \sqrt{-2 * log(u_1)} \cos(2\pi u_2) \tag{2.1}$$

$$Z_2 = \sqrt{-2 * log(u_1)} \sin(2\pi u_2) \tag{2.2}$$

Z_1 and Z_2 will be independent normally distributed random variables with mean 0 and standard deviation 1. Write a function to generate N normally distributed random numbers using the Box-Muller method.

2.2 Pricing Path-independent and Path-dependent options in the Black-Sholes model

Problem 1. Implement the class **BlackSholesModel** which contains as **private** member variables the initial price S_0, the drift μ, the volatility σ, the interest rate r and the current time t. The class should contain a constructor by default, a public function **GenerateRiskNeutralPricePath** (which generates a path under the risk neutral probability measure $\mathbb{Q}$) and a public function **GeneratePricePath** (which generates a path under the probability measure $\mathbb{P}$) (for more details, see the lecture).

Problem 2. Implement the class **PathIndependentOption** which contains as **private** member variables the maturity T and the strike K. The class should contain the following public functions:

- SetMaturity, GetMaturity, SetStrike, GetStrike;
- virtual double payoff(double x) const;
- virtual double price(const BlackSholesModel&) const;
- virtual destructor;

Write the derived classes **CallOption** and **PutOption**.

Problem 3. Write a class **MonteCarloPricer** which contains as **private member variables** the number of simulations and the number of time steps. The class should also contain a constructor by default, and the **public** function:

- double price(const BlackSholesModel&, const CallOption &) const;

Compute the price of the CallOption using the MonteCarloPricer and compare it with the price of the CallOption obtained by using the explicit formula.

Problem 4. Add a new function to **MonteCarloPricer** to price a **PutOption**. What aspects of your answer do you find unsatisfactory? How do you solve the problem?

Problem 5. Write **DigitalCallOption** and **DigitalPutOption** classes.

Problem 6. An up-and-out knock-out call option with strike K, barrier B, and maturity T is an option which pays off:

$$\begin{cases} \max\{S_T - K, 0\} & \text{if } S_t < B \text{ for all } t \in [0, T] \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

In other words, it has the same payoff as a call option with strike K unless the stock price hits the barrier level B before maturity, in which case it pays zero (in which one says it has knocked out).

A more practical contract is a discrete-time knock-out call option. This is essentially the same as the continuous-time version, except that you only test whether the stock price is below the barrier at some fixed time points.

Write a class **UpAndOutOption**.

Add a new function to **MonteCarloPricer** that prices discrete-time knowk-out options by simulating price paths of length **nSteps** and computing the payoff for these steps. By taking a large number of steps, the same function can be used to price continuous-time knock-out options. What aspects of your answer do you find unsatisfactory? How do you solve the problem?

Problem 7. Write an **AsianOption** which represents an Asian Option.

Problem 8. [**Computation of Greeks**] To compute the delta, Δ, of an option by Monte-Carlo, you can use the following algorithm.

- Choose a small value for h, say $h = S_0 \times 10^{-6}$.
- Generate N stock price paths.
- Use the Monte-Carlo method to compute the price of the option when the initial stock price is taken to be $S_0 + h$.
- Use the Monte Carlo method *with the same price paths* to compute the price of the option when the initial stock price is taken to be $S_0 - h$.
- Estimate the delta using the central difference estimate for the derivative.

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{2.4}$$

Compute the delta of an option using this algorithm. Give a test for your results.

Problem 9. [**Price standard deviation**] You can use the central limit theorem to estimate the standard deviation in our Monte Carlo estimate. In the Monte Carlo method, we have generated a sample of N discounted payoffs and taken the mean to compute an estimate for the true expectation. By the central limit theorem, this sample mean is assimptotically normally distributed about the true expectation with standard deviation $sN^{-\frac{1}{2}}$

where s is the standard deviation of the discounted payoff. We can estimate s by taking the standard deviation of our sample. Use this to compute a 95% confidence interval for Monte Carlo price. Add to the class **MonteCarlo pricer** a function **PriceWithSE** which returns an object containing both informations: the price and the standard deviation.

Problem 10. [Variance reduction] There are numerous *variance reduction techniques* that can be used to improve Monte-Carlo calculations. One approach is called *antithetic sampling*. We first simulate N stock price paths using the following $\mathbb{Q}$-measure model Equation:

$$\log(S_{t+\delta t}) = \log(S_t) + \left(\mu - \frac{1}{2}\sigma^2\right)\delta t + \sigma(\delta t)^{\frac{1}{2}} e_t \tag{2.5}$$

We then simulate N further stock price paths using the same random values e_t but now using the formula

$$\log(S_{t+\delta t}) = \log(S_t) + \left(\mu - \frac{1}{2}\sigma^2\right)\delta t - \sigma(\delta t)^{\frac{1}{2}} e_t \tag{2.6}$$

We can compute an estimate for the risk-neutral price by computing the discounted average payoff for all $2N$ paths. It turns out that for many options this gives a more accurate answer than one obtains by simulating $2N$ independent stock paths. Implement antithetic sampling to price a call option. Compute the standard deviation in this case.
You should consult the literature on Monte Carlo methods for more information on this and other variance reduction techniques.