1. **Crude Monte-Carlo and Antithetic Variates**

   (a) Implemente the crude Monte-Carlo algorithm to estimate the price of a European put option with parameters $S_0 = 100$, $r = 0.0175$, $\sigma = 0.4$, $T = 1$, $K = 40$. Use $N = 10^2, 10^3, 10^4, 10^5, 10^6$ normal random variables to generate the terminal asset price

   $$S_T^j = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z_j}$$

   needed. Report your estimates and a 95% confidence interval for each N. Compare your results to the Black-Scholes price.

   (b) Repeat part (a) using $N/2$ uniform random variables and anti-thetic sampling. What is the gain in efficiency over crude Monte-Carlo. (Ignore computation time).

2. **(Geometric Asian Option)** Suppose that the price of an asset at time t, S_t, follows the dynamics

   $$dS_t = rS_t dt + \sigma S_t dB_t$$

   under the risk-neutral measure Q (the Black-Scholes model). Consider a partition

   $$\pi = \{0 = t_0 < t_1 < \cdots < t_n = T\}$$

   of $[0, T]$ with equal spacing $\Delta t = (t_i - t_{i-1})$, $t_i = i\Delta t$, and $n\Delta t = T$. We have

   $$S_{t_i} = S_{t_{i-1}} e^{(r-\frac{1}{2}\sigma^2)\Delta t+\sigma(B_{t_i}-B_{t_{i-1}})}, \quad i = 1, 2, \ldots, n. \tag{1}$$

   The time-T expiry geometric average Asian call option with strike price K has payoff

   $$h_T = \left(\left[\prod_{i=1}^n S_{t_i}\right]^{\frac{1}{n}} - K\right)^+$$

   at time T.

   (a) Show that

   $$\begin{aligned}\log\left(\left[\prod_{i=1}^n S_{t_i}\right]^{\frac{1}{n}}\right) - \log(S_0) &= \sum_{i=1}^n \frac{i}{n}\left[(r-\frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}Z_{n-i+1}\right] \\ &= \frac{(n+1)}{2}(r-\frac{1}{2}\sigma^2)\Delta t + \sum_{i=1}^n \sigma_i Z_i\end{aligned}$$

   where $\{Z_i\}_{i=1}^n$ are i.i.d. standard normal random variables and $\sigma_i = \frac{i}{n}\sigma\sqrt{\Delta t}$.

   (b) Show that $\log\left(\left[\prod_{i=1}^n S_{t_i}\right]^{\frac{1}{n}}\right) - \log(S_0)$ has normal distribution with mean

   $$(r-\frac{1}{2}\sigma^2)\frac{(n+1)}{2n}T$$

   and variance

   $$\frac{\sigma^2(n+1)(2n+1)}{6n^2}T.$$

(c) Show that the time-zero price of the time-T expiry geometric average Asian call option with strike price K is

$$C_0^{GA} = e^{-rT}\left(S_0 e^{\hat{\mu}T}\Phi(\hat{d}_1) - K\Phi(\hat{d}_2)\right)$$

where

$$\hat{\mu} = (r - \frac{1}{2}\sigma^2)\frac{(n+1)}{2n} + \frac{1}{2}\hat{\sigma}^2, \qquad \hat{\sigma}^2 = \frac{\sigma^2(n+1)(2n+1)}{6n^2}$$
$$\hat{d}_1 = \frac{\log(S_0/K) + (\hat{\mu} + \frac{1}{2}\hat{\sigma}^2)T}{\hat{\sigma}\sqrt{T}}, \qquad \hat{d}_2 = \hat{d}_1 - \hat{\sigma}\sqrt{T}.$$

3. **(Arithmetic Asian Option)** The time-T expiry arithmetic average Asian call option with strike price K has payoff

$$h_T = \left(\frac{1}{n}\sum_{i=1}^{n} S_{t_i} - K\right)^+$$

at time T. Under the assumptions of the previous question we wish to calculate the time-zero arithmetic Asian option price by Monte-Carlo simulation as there is no closed form formula.

(a) Suppose $r = 0.0175$, $\sigma = 0.25$, $S_0 = 500$, $K = S_0$, $T = 1$, and $n = 52$. Write a program that generates a path

$$(S_0, S_{t_1}, \dots, S_{t_n})$$

of the asset price using equation (1).

(b) Write a program that uses N-paths of the asset price to calculate the geometric and arithmetic average Asian call option prices at time zero using (crude) Monte-Carlo. Compare your results to the analytic formula results of Question 2 for $N = 10^2, 10^3, 10^4, 10^5, 10^6$. Report 95% confidence intervals for your Monte-Carlo estimates for each N.

(c) **(Antithetic Variates)** Modify your code from part (b) to reduce the variance of the Monte-Carlo estimate of the arithmetic average Asian call option price by using the anti-thetic variates. Report the estimated price, variance of the estimate, and efficiency compared to the crude Monte-Carlo estimate from part (b) for the each value N as in part (b).

(d) **(Control Variates)** Modify your code from part (b) to reduce the variance of the Monte-Carlo estimate of the arithmetic average Asian call option price by using the geometric average Asian call option as a control variate. Recall that the control variate Monte-Carlo estimator for $E[f(\vec{U})]$ is

$$\hat{\theta}_{cv} = \frac{1}{M}\sum_{i=1}^{M}\{f(\vec{U}_i) - c\left(g(\vec{U}_i) - E[g(\vec{U}_i)]\right)\}.$$

The optimal c is first estimated using pilot m uniform random vectors $\hat{U}_j$ as

$$\hat{c} = \frac{\sum_{i=1}^{m} f(\vec{U}_i)g(\vec{U}_i) - m\hat{\theta}_{CR}\hat{\theta}_g}{(m-1)\hat{S}_g^2},$$

where

$$\hat{\theta}_{CR} = \frac{1}{m}\sum_{i=1}^{m} f(\vec{U}_i),$$
$$\hat{\theta}_g = \frac{1}{m}\sum_{i=1}^{m} g(\vec{U}_i), \quad \text{and}$$
$$\hat{S}_g^2 = \frac{1}{m-1}\sum_{i=1}^{m}(g(\vec{U}_i) - \hat{\theta}_g)^2.$$

Here f is the discounted payoff of the arithmetic average Asian call option, g is the discounted payoff of the geometric average Asian call option, and each $\vec{U}_i$ is a vector of $n = 52$ independent $U(0,1)$ random variables used to generate the one path of the stock price. Report the estimated price, variance of the estimate, and efficiency compared to the crude Monte-Carlo estimate from part (b) for the each value N as in part (b) with $m = 0.1 \times N$ and $M = (N - m)$.

4. **(Heston Stochastic Volatility Model)** Suppose that the joint dynamics of the asset price and volatility are given by

$$dS_t = rS_t dt + \sqrt{v_t}S_t[\rho dB_t^{(1)} + \sqrt{1-\rho^2}dB_t^{(2)}] \tag{2}$$
$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t}dB_t^{(1)} \tag{3}$$

where $B_t^{(1)}$ and $B_t^{(2)}$ are independent standard Brownian motions under the risk-neutral probability measure. Here v_t is the variance and $\sqrt{(v(t))}$ is the stochastic volatility. Note θ is the long-run average volatility (mean-reversion level), κ is the *speed of mean-reversion*, and σ is the *volatility of volatility*. In order to ensure that v_t remains positive we require $2\kappa\theta \geq \sigma^2$.

(a) **(Euler-discretization)** Consider the discretization of equations (2)-(3), with $x_t = \log(S_t)$,

$$x_{t_i} \approx x_{t_{i-1}} + (r - \frac{1}{2}v_{t_{i-1}})\Delta t + \sqrt{v_{t_{i-1}}}[\rho(B_{t_i}^{(1)} - B_{t_{i-1}}^{(1)}) + \sqrt{1-\rho^2}(B_{t_i}^{(2)} - B_{t_{i-1}}^{(2)})] \tag{4}$$
$$v_{t_i} \approx v_{t_{i-1}} + \kappa(\theta - v_{t_{i-1}})\Delta t + \sigma\sqrt{v_{t_{i-1}}}(B_{t_i}^{(1)} - B_{t_{i-1}}^{(1)}) \tag{5}$$

for $i = 1, \ldots, n$ and $\Delta t = t_i - t_{i-1}$ a partition of $[0, T]$ as in Question 2.

- Write a program which; given initial values S_0, v_0, and the parameters $(r, \rho, \kappa, \theta, \sigma)$; generates a path of the joint stock and volatility process using equations (4)-(5).
- In order to ensure that $v_t \geq 0$, which is not guaranteed by the Euler discretization, we impose the condition that if $v_{t_{i-1}} < 0$ we set $v_{t_{i-1}} = 0$. Be careful to write your algorithm in such a way that this condition is implemented and you don't end up taking the square root of a negative number in either of equations (4) or (5).
- Test your program and plot representative paths of the stock price and volatility processes for the the parameter sets below

$r = 0$	$\rho = -0.7$	$\kappa = 2$	$\theta = 0.03$	$\sigma = 0.1$
$r = 0$	$\rho = 0.0$	$\kappa = 2$	$\theta = 0.03$	$\sigma = 0.1$
$r = 0$	$\rho = +0.7$	$\kappa = 2$	$\theta = 0.03$	$\sigma = 0.1$

with $S(0) = 100$, $v_0 = 0.1$, $T = 0.5$, $n = 125$.

- Write a program to estimate the time-zero price of an at the money European call option expiring at $T = 0.5$ using the crude Monte-Carlo estimator

$$\hat{C}_0 = \frac{1}{N}\sum_{j=1}^{N} e^{-rT}(S_T^j - K)^+$$

where S_T^j is a simulated value of the time-T price of the asset generated using the Euler discretization (note $S_T = e^{x_n}$). Report your estimates a 95% confidence interval for $N = 10^2, 10^3, 10^4, 10^5$.

(b) **(Conditional Monte-Carlo)** It can be shown that conditional on the path of the volatility process $\{v_s : 0 \le s \le T\}$ the log asset price $\log(S_T)$ is a normal random variable with mean a and variance b with

$$a = \log(S_0)\xi + rT - \frac{1-\rho^2}{2}\int_0^T v_t\,dt$$
$$b = (1-\rho^2)\int_0^T v_t\,dt$$

and where

$$\xi = \exp\left(-\frac{\rho^2}{2}\int_0^T v_t\,dt + \rho\int_0^T \sqrt{v_t}\,dB_t^{(1)}\right).$$

Then, conditional on the path of $\{B_t^{(1)} : 0 \le t \le T\}$,

$$E^Q[e^{-rT}(S_T - K)^+ | \{B_t^{(1)} : 0 \le t \le T\}] = S_0\xi\Phi(\tilde{d}_1) - Ke^{-rT}\Phi(\tilde{d}_2),$$

where

$$\tilde{d}_1 = \frac{\log(S_0\xi/K) + (r + \frac{1}{2}\tilde{\sigma}^2(1-\rho^2))T}{\tilde{\sigma}\sqrt{T(1-\rho^2)}}$$
$$\tilde{d}_2 = \tilde{d}_1 - \tilde{\sigma}\sqrt{T(1-\rho^2)}$$
$$\tilde{\sigma}^2 = \frac{1}{T}\int_0^T v_t\,dt.$$

Hence we can apply conditional Monte-Carlo as follows.

- Approximate $\tilde{\sigma}^2$ by

$$\tilde{\sigma}^2 \approx \frac{1}{T}\sum_{i=1}^{n} v_{t_{i-1}}\Delta t$$

- Approximate ξ by

$$\xi \approx \exp\left(-\frac{\rho^2}{2}\sum_{i=1}^{n} v_{t_{i-1}}\Delta t + \rho\sum_{i=1}^{n}\sqrt{v_{t_{i-1}}}(B_{t_i}^{(1)} - B_{t_{i-1}}^{(1)})\right).$$

- Use the (approximate) conditional Monte-Carlo estimator of the time-zero price of the call option

$$C_0^{cmc} = \frac{1}{N}\sum_{j=1}^{N}[S_0\xi^j\Phi(\tilde{d}_1^j) - Ke^{-rT}\Phi(\tilde{d}_2^j)]$$

where ξ^j, $\tilde{d}_1^j$, and $\tilde{d}_2^j$ are simulated values. Hence with conditional Monte-Carlo we only need to simulate paths of the volatility process $\{v_{t_i} : i = 1, \ldots n\}$ and not $\{S_{t_i} : i = 1, \ldots n\}$.

Implement this CMC estimate using the same parameter values and values of N given in part (a). Report the estimated call price and the 95% confidence interval in each case. Report the efficiency of using CMC compared to crude Monte-Carlo in each case.

A Random Number Generation

You will need to be able to generate large numbers of pseudo-random numbers in this assignment. The appendix is included to help you create code to generate pseudorandom numbers in `C++`. There are no problems to be graded in this section. If you are doing this in `C++` I recommend you use the *multiple recursive generator* MRG32k3a of [3]:

`http://www.iro.umontreal.ca/~lecuyer/myftp/papers/combmrg2.ps`

The algorithm is given by

$$\begin{aligned}
x_{1,i} &= (1403580 x_{1,i-2} - 810728 x_{1,i-3}) \mod (2^{32} - 209) \\
x_{2,i} &= 527612 x_{2,i-1} - 1370589 x_{2,i-3} \mod (2^{32} - 22853) \\
z_i &= (x_{1,i} - x_{2,i}) \mod (2^{32} - 209) \\
u_i &= z_i/(2^{32} - 209)
\end{aligned}$$

This generator has period of 2^{191}. To generate n pseudo-random $U(0,1)$ numbers $u_1, \ldots, u_n$ The algorithm is initialized with seed

$$\begin{aligned}
x_{1,-1} = 56789, \quad x_{1,-2} = 12345, \quad x_{1,0} = 89012, \\
x_{2,0} = 32109, \quad x_{2,-2} = 98765, \quad x_{2,-1} = 54321.
\end{aligned}$$

You may find it helpful to look at the `C` code for algorithm at

`http://www.iro.umontreal.ca/~lecuyer/myftp/papers/combmrg2.c`

which implements a few features to ensure we don't ever return $u_i = 0$ or $u_i = 1$. **If you attempt to implement the code in a language such as R you may encounter difficulties or get poor results as the way floating point numbers are handled in R is generally not sufficient for an accurate implementation.**

1. Implement the above algorithm and report the first 10 numbers $u_1, \ldots, u_{10}$ generated as well as $\sum_{i=1}^{10} u_i$. You should get the following results

u_1	0.127011
u_2	0.318528
u_3	0.309186
u_4	0.825847
u_5	0.221630
u_6	0.533395
u_7	0.480774
u_8	0.355560
u_9	0.135988
u_{10}	0.755852
$\sum_{i=1}^{10} u_i$	4.063772

In your implementation it is helpful to be able to start the algorithm from where it left off when last used by saving the last values necessary to continue.

2. In order to generate standard normal random variables we shall use the inversion method with the approximation of the inverse of the standard normal cumulative distribution function $\Phi^{-1}(u)$ given by

$$\Phi^{-1}(u) \approx \frac{\sum_{n=0}^{3} a_n(u-\frac{1}{2})^{2n+1}}{1+\sum_{n=0}^{3} b_n(u-\frac{1}{2})^{2n+2}}$$

for $0.5 \leq u \leq 0.92$ and

$$\Phi^{-1}(u) \approx \sum_{n=0}^{8} c_n[\log(-\log(1-u))]^n$$

for $0.92 \leq u < 1$. If we have $0 < u < 0.5$ then we use the symmetry of the distribution so that

$$\Phi^{-1}(u) = -\Phi^{-1}(1-u)$$

and the case applicable for $(1-u)$ (between 0.5 and 0.92 or greater than 0.92). The constants are:

$a_0 = 2.50662823884$	$c_0 = 0.3374754822726147$
$a_1 = -18.61500062529$	$c_1 = 0.9761690190917186$
$a_2 = 41.39119773534$	$c_2 = 0.1607979714918209$
$a_3 = -25.44106049637$	$c_3 = 0.0276438810333863$
$b_0 = -8.47351093090$	$c_4 = 0.0038405729373609$
$b_1 = 23.08336743743$	$c_5 = 0.0003951896511919$
$b_2 = -21.06224101826$	$c_6 = 0.0000321767881768$
$b_3 = 3.13082909833$	$c_7 = 0.0000002888167364$
	$c_8 = 0.0000003960315187$

This is the algorithm discussed in [1] and the algorithm is implemented in C++ in the file `Normals.cpp` from [2].

- Implement the inversion algorithm so that you can generate n independent standard normal pseudo-random numbers given a vector of n pseudo-random numbers on (0,1).

References

[1] Paul Glasserman. *Monte Carlo methods in financial engineering*. Springer-Verlag, New York, 2004.

[2] Mark Joshi. *C++ design patterns and derivatives pricing*. Cambridge University Press, Cambridge, 2004.

[3] Pierre L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1):pp. 159–164, 1999.

Email me if you find any typos or have any problems.