Math 16:642:623 – Homework Assignment 3* – Spring 2010 © Paul Feehan

Practice or *advanced* problems are optional and are not graded: *practice problems* are intended as drill problems and aides to exam preparation while *advanced problems* are intended for students with additional mathematics background. Please consult the *Homework Submission Requirements* before commencing work on this assignment.

1. Reading Assignments and Sample Code

1.1. **Reading assignments.** The primary reading assignments for this homework set are

- Glasserman, chapter 2.
- Joshi, chapters 7 and 9.

You may find the following suggested alternative readings helpful:

- Clewlow and Strickland, sections 4.11 and 4.12.
- Jäckel, sections 2.4, 10.1, and 10.3.
- Knuth, *Art of Computer Programming*, Addison-Wesley, 1998; Volume II, chapter 3.
- London, chapter 2.
- Press et al., chapter 7.
- Seydel, chapter 2.
- Section 17.9, "Random number generator algorithms" and chapter 19, "Random Number Distributions", in the GNU Scientific Library manual:
  `www.gnu.org/software/gsl/manual/html_node/index.html`

2. Programming and Written Assignments

1. Write a C++ program to price a European-style call option using the following method and test data. Assume that the stock price process is geometric Brownian motion with volatility $\sigma = 0.3$, initial asset price $S(0) = 100$, constant risk-free interest rate $r = 0.05$, option type call, strike $K = 110$, and maturity $T = 1$ year; for the Monte Carlo simulations, use $J = 10,000$ paths.

(a) *L'Ecuyer uniform random number generator.* Write a C++ program to implement the L'Ecuyer uniform random number generator using a combination of two multiple recursive generators; use a seed $(x_{10}, x_{11}, x_{12}, x_{20}, x_{21}, x_{22}) = (1, 2, 3, 5, 7, 11)$. Develop and test your code, `LEcuyer.h` and `LEcuyer.cpp`, following the model of `ParkMiller.h` and `ParkMiller.cpp` in section 6.4 of Joshi. Modify `RandomMain3.cpp` in section 6.6 of Joshi to allow Monte Carlo pricing by both the Park-Miller and the L'Ecuyer generators and by the Black-Scholes-Merton formula. Your submitted program should output the results as

```
Closed-form option price =
MC option price with Park-Miller uniform generator =
MC option price with L'Ecuyer uniform generator =
```

While C code for the L'Ecuyer algorithm is publicly available (for example, Figure 2.3 in Glasserman), you should code your own algorithm.

(b) *Fishman acceptance-rejection algorithm.* Write a C++ program to implement the Fishman (1996) method to generate samples from the standard normal distribution using the acceptance-rejection algorithm (see notes below for the algorithm). Note that the Marsaglia-Bray modification of Box-Müller algorithm is implemented in `Random1.h` and `Random1.cpp` (see Joshi, section 1.3), while the inverse cumulative distribution function method is implemented in `Random2.h` and `Random2.cpp`. Develop your code as `Random4.h` and `Random4.cpp`, to replace the inverse cumulative distribution function method of `Random2.h` and `Random2.cpp`. As in (a), modify `RandomMain3.cpp` to allow Monte Carlo pricing by the Black-Scholes-Merton formula using the default inverse cumulative distribution function, Marsaglia-Bray

*Last update: February 2, 2010 7:32

modified Box-Müller, and the Fishman generators. Your submitted program (using the default Park-Miller algorithm to generate samples from the uniform distribution, so modify `Random1.h` and `Random1.cpp` accordingly) should output the results as

```
MC option price with inverse distribution normal generator =
MC option price with Box-Muller normal generator =
MC option price with Fishman normal generator =
```

2. Use your C++ program to generate data for plots of the probability density functions determined by the sequences of normal random variables generated by the Marsaglia-Bray-Box-Müller, inverse transform, or Fishman algorithms, when driven by the Park-Miller or L'Ecuyer uniform random number generators.

(c) *Program submission.* Your submitted code file should only include relevant code from `LEcuyer.cpp`, `Random1.cpp`, `Random4.cpp`, `LEcuyer.h`, `Random1.h`, `Random4.h`, and your modified main program `RandomMain3.cpp`; the remaining files required by `RandomMain3.cpp` (that is, `AntiThetic.cpp, Arrays.cpp, ConvergenceTable.cpp, MCStatistics.cpp, Normals.cpp, Parameters.cpp, ParkMiller.cpp, PayOff3.cpp, PayOffBridge.cpp, Random2.cpp, SimpleMC8.cpp Vanilla3.cpp`) can be assumed to be includable using from the directory or folder containing `RandomMain3.cpp` using, for example:

```
#include<AntiThetic.h>
```

The lines `double tmp; cin >> tmp;` should also be removed from `RandomMain3.cpp`.

(d) Write a brief report with a short explanation of the algorithms and their implementation and an analysis of your results.

Your report should include plots (for example, with MATLAB, Excel, or Gnuplot) comparing the graph of the standard normal probability density function (curve plot of $y = e^{-x^2/2}/\sqrt{2\pi}$, $x \in [-4, 4]$) with each one of the 6 methods mentioned in this assignment for simulating samples from the normal probability density (see `http://en.wikipedia.org/wiki/Normal_distribution`). Each plot should have a graph of the standard normal probability density function superimposed with a plot of points y obtained from a histogram for one approximation method with 1,000 samples, bin width 0.05, and $x \in [-4, 4]$. For more details on histograms, see `http://en.wikipedia.org/wiki/Histogram`.

3. Practice Exercises

3. In order to model the stochastic evolution of the recovery rate of a non-performing loan one may use the following model for the recovery rate, $R(t) \in [0, 1]$:

$$dR(t) = \kappa(1 - R(t) - \theta)\, dt + \sigma\sqrt{R(t)(1 - R(t))}\, dW(t).$$

(a) What is the Kolmogorov forward equation for the transition probability density, $p(t, x; T, y)$, for $R(T)$ in the y variable, given $R(t) = x$?

(b) Find the partial differential equation satisfied by the stationary density,

$$p(y) = \lim_{T\to\infty} p(0, x; T, y).$$

(c) Show that $p(y) = f_{\alpha,\beta}(y)$, the Beta density with parameters

$$\alpha = \frac{2\kappa\theta}{\sigma^2}, \quad \beta = \frac{2\kappa(1-\theta)}{\sigma^2}.$$

4. How can you test the quality your random number generators? See Knuth for a survey of tests.

5. Prove that the acceptance-rejection algorithm generates a sequence of samples from a desired target probability distribution, given an algorithm for generating a sequence of samples from the uniform distribution. See section 2.2.2. in Glasserman for details.

6. Use *floating point arithmetic* (see, for example, Figure 2.4 in Glasserman) to implement the linear congruential generator with the Mersenne prime, $m = 2^{19,937} - 1$, and a suitable multiplier a (see the 1997 and 1998 papers of Matsumoto-Nishimura) to build a uniform random number generator. Compare the computation time required by the Matsumoto-Nishimura generator with that of L'Ecuyer to generate a sequence of 10^6 uniform random numbers. The is implemented in GSL as `gsl_rng_mt19937`; see section 17.9, "Random number generator algorithms", in the GNU Scientific Library manual and section 7.5 in Jaeckel.

7. Write a C++ program to price each of the following path-dependent European-style equity options using Monte Carlo and, where a formula is available, closed-form: *single barrier* call or put, where the barrier may be in or out, and a *double barrier* call or put, where each barrier may be in or out (eight double barrier types). Compare your Monte Carlo results with the closed-form results. Closed form formulae are available for all single barrier options in Haug (section 4.17.1) and Wilmott (chapter 23) and for up-and-out-and-down-and-out calls and puts in Haug (section 4.17.3). Derive a formula for one of the other styles of double-barrier options, for example, an up-and-in-and-down-and-out call, by modifying the Fourier series solution to the heat equation on a finite interval in the real line.

8. Write a C++ program using the Brownian bridge to generate paths $W(t_i)$, for $i = 1, \ldots, I$, where the integer I is a power of 2, to obtain approximate solutions, $S^j(t_i)$, $i = 1, \ldots, I$, $j = 1, \ldots, J$, to $dS(t) = S(t)((r - d)\,dt + \sigma\,dW(t))$, and hence compute the call and put option prices. See Jäckel, section 10.8.3 and Glasserman, section 3.1, for details. If Σ denotes the covariance matrix of $W(t_1), \ldots, W(t_I)$, implement the Cholesky decomposition algorithm described in Glasserman, pp. 72-73, to find A such that $AA^T = \Sigma$, and hence generate vector samples from the multivariate normal distribution of $W(t_1), \ldots, W(t_I)$.

9. Write a C++ program to generate sequences of uniform random variables using one or more of the `gsl` functions described in section 17.9, "Random number generator algorithms", in the GNU Scientific Library manual. For each generator, use MATLAB, Excel, or Gnuplot to plot 1,000 pairs of uniform samples on the square $[0, 1] \times [0, 1]$.

4. Supplementary Notes

4.1. **Implementing linear congruential generators in integer arithmetic without overflow.** To implement a linear congruential generator, $y_{i+1} = ay_i \mod m$ in integer arithmetic without overflow, apply the method of section 2.1.3 in Glasserman, which applies when $r \leq q$, where $q = \lfloor m/a \rfloor$ (greatest integer in m/a) and $r = m \mod a$, so that $m = aq + r$:

- $h \leftarrow \lfloor y/q \rfloor$;
- $y \leftarrow a * (y - h * q) - h * r$;
- if $(y < 0)$ $y \leftarrow y + m$;
- if $(y \equiv 0)$ $y \leftarrow m$;
- $u \leftarrow y * k$;

where, in the final step, $k = 1/(m + 1)$. The algorithm ensures that every intermediate step in the calculation of $ay_i \mod m$ results in an integer between $-(m - 1)$ and $m - 1$. The Park-Miller algorithm uses this method with $m = 2^{31} = 2147483647$, $a = 39373$, $q = 127773$, and $r = 2836$.

4.2. **L'Ecuyer method of generating samples from the uniform distribution.** See Example 4 in L'Ecuyer, *Combined recursive multiple random number generators*, Operations Research **44** (1996), pp. 816-822. Set $m_1 = 2^{31} - 1 = 2147483647$, $a_{11} = 0$, $a_{12} = 63308$, $a_{13} = -183326$, and $m_2 = 2145483479$, $a_{21} = 86098$, $a_{22} = 0$, $a_{23} = -539608$. Note that $m_1 > m_2$. Then for any integer $i \geq 3$, define

$$\begin{aligned} x_{1,i} &= (a_{11}x_{1,i-1} + a_{12}x_{1,i-2} + a_{13}x_{1,i-3}) \mod m_1, \\ x_{2,i} &= (a_{21}x_{2,i-1} + a_{22}x_{2,i-2} + a_{23}x_{2,i-3}) \mod m_2, \\ x_i &= x_{1,i} - x_{2,i} \mod m_1, \\ u_i &= x_i/m_1. \end{aligned}$$

Integer seed values must be supplied for $x_{1,2}, x_{1,1}, x_{1,0}$ and $x_{2,2}, x_{2,1}, x_{2,0}$. The generator has a period of approximately 2^{185} and is implemented in GSL as `gsl_rng_cmrg`. See section 17.9, "Random number generator algorithms", in the GNU Scientific Library manual, the paper by L'Ecuyer, and section 2.1.5 in Glasserman for further details.

To implement the algorithm, observe that

$$\begin{aligned} x_{1,i} &= ((a_{11}x_{1,i-1}) \mod m_1 + (a_{12}x_{1,i-2}) \mod m_1 + (a_{13}x_{1,i-3}) \mod m_1) \mod m_1, \\ x_{2,i} &= ((a_{21}x_{2,i-1}) \mod m_2 + (a_{22}x_{2,i-2}) \mod m_2 + (a_{23}x_{2,i-3}) \mod m_2) \mod m_2, \end{aligned}$$

and apply the method of computing $ax \mod m$, in integer arithmetic without overflow, described in the preceding section. Thus one needs to compute

$$\begin{aligned} x_{13} &= (a_{12}x_{11} + a_{13}x_{10}) \mod m_1, \\ x_{23} &= (a_{21}x_{22} + a_{23}x_{20}) \mod m_2. \end{aligned}$$

A C implementation of the above algorithm can be found in the article by L'Ecuyer (p. 821) or Glasserman (Figure 2.3). A similar pseudocode algorithm is provided below, which provides one sample, u_1, from the uniform distribution given seed values, x_{12}, x_{11}, x_{10} and x_{22}, x_{21}, x_{20}:

- Precompute $q_{12}, q_{13}, q_{21}, q_{23}$, where $q_{12} = \lfloor m_1/a_{12} \rfloor$, etc., and precompute $r_{12}, r_{13}, r_{21}, r_{23}$, where $r_{12} = m_1 - a_{12}q_{12}$, etc., and $k = 1/(m + 1)$;
- // Compute $a_{12} * x_{11} \mod m_1$:
- $h \leftarrow \lfloor x_{11}/q_{12} \rfloor$;
- $p_{12} \leftarrow a_{12} * (x_{11} - h * q_{12}) - h * r_{12}$;
- if $(p_{12} < 0)$ $p_{12} \leftarrow p_{12} + m_1$;
- // Compute $a_{13} * x_{10} \mod m_1$:
- $h \leftarrow \lfloor x_{10}/q_{13} \rfloor$;
- $p_{13} \leftarrow a_{13} * (x_{10} - h * q_{13}) - h * r_{13}$;
- if $(p_{13} < 0)$ $p_{13} \leftarrow p_{13} + m_1$;
- // Compute $a_{21} * x_{22} \mod m_2$:

- $h \leftarrow \lfloor x_{22}/q_{21} \rfloor$;
- $p_{21} \leftarrow a_{21} * (x_{22} - h * q_{21}) - h * r_{21}$;
- if $(p_{21} < 0)$ $p_{21} \leftarrow p_{21} + m_2$;
- // Compute $a_{23} * x_{20} \mod m_2$:
- $h \leftarrow \lfloor x_{20}/q_{23} \rfloor$;
- $p_{23} \leftarrow a_{23} * (x_{20} - h * q_{23}) - h * r_{23}$;
- if $(p_{23} < 0)$ $p_{23} \leftarrow p_{23} + m_2$;
- // Update x_{11} and x_{10}:
- $x_{10} \leftarrow x_{11}$;
- $x_{11} \leftarrow x_{12}$;
- // Compute $(p_{12} + p_{13}) \mod m_1$ and update x_{12}:
- $p_{12} \leftarrow p_{12} - m_1$;
- $x_{12} \leftarrow p_{12} + p_{13}$;
- if $(x_{12} < 0)$ $x_{12} \leftarrow x_{12} + m_1$;
- // Update x_{21} and x_{20}:
- $x_{20} \leftarrow x_{21}$;
- $x_{21} \leftarrow x_{22}$;
- // Compute $(p_{21} + p_{23}) \mod m_2$ and update x_{22}:
- $p_{21} \leftarrow p_{21} - m_2$;
- $x_{22} \leftarrow p_{22} + p_{23}$;
- if $(x_{22} < 0)$ $x_{22} \leftarrow x_{22} + m_2$;
- // Compute $x_1 \mod m_1$:
- if $(x_{12} < x_{22})$ $x_1 \leftarrow x_{12} - x_{22} + m_1$;
- else $x_1 \leftarrow x_{12} - x_{22}$;
- // Compute u_1:
- if $(x_1 \equiv 0)$ $x_1 \leftarrow m_1$;
- $u_1 \leftarrow x_1 * k$;

The choice of k and the second last line ensure that $u_1 \in (0, 1)$ rather than $u_1 \in [0, 1]$; this avoids problems when applying the inverse cumulative distribution function method when $u_1 = 0$ or 1.

4.3. **Fishman method of generating samples from the standard normal distribution.** See Glasserman, section 2.2.2, for a description of the acceptance-rejection algorithm. Choose $g(x) = \exp(-|x|)/2$, $x \in (-\infty, \infty)$ (the double exponential probability density function) and $f(x) = \exp(-x^2/2)/\sqrt{2\pi}$, $x \in (-\infty, \infty)$ (the standard normal probability density function).

To generate samples from the double exponential probability distribution, we first modify the inverse transform method of Example 2.2.1 in Glasserman, which applies to the exponential probability density function. Let $G(x)$ denote the cumulative double exponential probability distribution function,

$$G(x) = \int_{-\infty}^{x} g(y)\, dy = \begin{cases} \frac{1}{2}e^x, & \text{if } x \le 0, \\ 1 - \frac{1}{2}e^{-x}, & \text{if } x \ge 0. \end{cases}$$

Setting $u = G(x)$ and inverting, we see that

$$x = \begin{cases} \log(2u), & \text{if } 0 < u \le \frac{1}{2}, \\ -\log(2(1-u)), & \text{if } \frac{1}{2} \le u < 1. \end{cases}$$

Since $2U \sim \text{Unif}[0, 1] \iff U \sim \text{Unif}[0, \frac{1}{2}]$ and $U \sim \text{Unif}[\frac{1}{2}, 1] \iff 2(1 - U) \sim \text{Unif}[0, 1]$, we can generate samples from the double exponential probability distribution using

$$X = \pm \log U,$$

where $U \sim \text{Unif}[0, 1]$ and the positive sign is selected with probability $\frac{1}{2}$, that is, we generate

$$X = \log U,$$

and randomize the sign of X.

For the given functions $f(x)$ and $g(x)$, we have

$$\frac{f(x)}{g(x)} = \sqrt{\frac{2}{\pi}} e^{-\frac{1}{2}x^2+|x|} \leq \frac{2e}{\pi} \approx 1.3155 =: c.$$

Therefore, $f(x) \leq cg(x)$, $x \in \mathbb{R}$. The rejection test $u > f(x)/cg(x)$ can be implemented as

$$u > \exp\left(-\frac{1}{2}x^2 + |x| - \frac{1}{2}\right) = \exp\left(\frac{1}{2}(|x|-1)^2\right).$$

Because $f(-x) = f(x)$ and $g(-x) = g(x)$, it suffices to generate positive samples and randomize the sign of X only if the sample is accepted. The combined algorithm is

- Generate U_1, U_2, U_3 from Unif$[0,1]$;
- $X \leftarrow -\log U_1$;
- If $U_2 > \exp(-0.5(X-1)^2)$, then go to Step 1;
- If $U_3 \leq 0.5$, then $X \leftarrow -X$;
- Return X;

See Example 2.2.7 in Glasserman for additional details.