Math G4077 – Homework Assignment 3* – Fall 2012 © Paul Feehan

1. Reading Assignments and Sample Code

1.1. **Reading assignments.** The primary texts and readings for the course are Glasserman, Joshi, Wilmott, and Achdou and Pironneau. You may find the suggested alternative readings helpful. The primary reading assignments for this homework set are

- Glasserman, sections 2.1.1, 3.1, 4.3, 5.1.1, 5.1.3, 5.5, and 5.6.
- Joshi, chapters 2, 3, and 7.

You may find the following suggested complementary readings helpful:

- Brandimarte, sections 8.1 and 8.4.
- Clewlow and Strickland, section 4.12.
- Haug, sections 8.3 and 8.4.4.
- Jäckel, chapter 8, sections 10.4, 10.8, 10.9.
- Jackson and Staunton, chapter 12.
- London, section 2.4.
- Press et al., sections 7.8 and 7.9.
- Seydel, section 2.5.
- Tavella, pages 93-100 and 102-113.
- Chapter 18, "Quasi-random sequences", in the GNU Scientific Library manual:
  `www.gnu.org/software/gsl/manual/html_node/index.html`
- Wikipedia — Low-discrepancy sequences:
  `en.wikipedia.org/wiki/Low-discrepancy_sequence`
- Wikipedia — Monte Carlo methods in finance:
  `en.wikipedia.org/wiki/Monte_Carlo_methods_in_finance`
- Wikipedia — Monte Carlo methods for option pricing:
  `en.wikipedia.org/wiki/Monte_Carlo_methods_for_option_pricing`
- Wikipedia — Sobol sequence:
  `en.wikipedia.org/wiki/Sobol_sequence`

2. Programming and Written Assignments

Problem 2.1. This is *part one* of a two-part assignment to write a C++ program to price European-style, discretely-sampled, arithmetic and geometric Asian call (or put) options using the following test data and the methods described below. 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 $r = 0.05$, zero dividend yield $q = 0$, option type is a European-style geometric Asian call (or put) discretely sampled at dates $t_1, \ldots, t_d$ (where $0 = t_0 < t_1 < \cdots < t_d = T$, using $d = 12$ for Asian call or put options and $d = 1$ for vanilla call or put options), strike $K = 110$, and maturity $T = 1$ year; for the Monte Carlo simulations, use $10,000$ paths.

Important: I provide specific **suggestions** for how to develop a C++ program using sample code provided by Joshi for the Monte Carlo and tree part of the course, **but** you may program the solutions however you wish using C++, **or** adapt MATLAB code of Brandimarte, **or** develop your own program from scratch however you wish, using C++, C, or MATLAB. (If you wish to use another language, please check with the teaching assistants first.) However you program the solutions, I recommend that that you use good object-oriented design (possible even using

*Last update: September 26, 2012 14:19

MATLAB) and avoid the temptation to simply "borrow" code from other sources; if you have not taken a course which covers object-oriented design, you may ignore this recommendation. The final projects will be designed so as to ensure that you need to write a substantial part of your own code. The remainder of the assignment refers to C++ but again, please **remember** the preceding comments.

Part *one* of the Asian options exercise (this assignment) asks you to allow use of the Sobol sequence generator as well as the Park-Miller pseudo-random number generator *to price a European-style vanilla call*; part *two* of the exercise (the next assignment) asks you to modify Joshi's code to allow path generation for the lognormal spot process using the Brownian bridge, allow geometric as well as arithmetic Asian call options, and may ask for implementation of another variance reduction method (such as stratified or importance sampling) when using Monte Carlo (but not quasi-Monte Carlo).

(a) You may create your program by either modifying Joshi's main program `RandomMain3.cpp` and include files from chapter 6 (simpler code but may require more work to modify in order to price the path-dependent arithmetic Asian option) or `EquityFXMain.cpp` and include files from chapter 7 (more complex code incorporating classes tailored for path-dependent options, but readily adaptable to price different path-dependent, single-asset products); `EquityFXMain.cpp` is configured to price an *arithmetic* Asian option (no closed-form formula), whereas Homework 5 ultimately asks you to price both *geometric* and *arithmetic* Asian options (closed-form formula easily derived for geometric Asian options contingent on a geometric Brownian motion spot process).

(b) In your program, replace calls in Joshi's code to the Park-Miller uniform random number generator (`ParkMiller.h` and `ParkMiller.cpp`) with calls to a d-dimensional Sobol sequence generator (`Sobol.h` and `Sobol.cpp`) in `RandomMain3.cpp` (or `EquityFXMain.cpp`). Antithetic sampling should be turned off (see `Antithetic.h` and `Antithetic.cpp`), as that concept of "variance reduction" is not appropriate for quasi-random sequences. Compare the price of a European-style call option (with the given test data) for the Sobol and Park-Miller (with and without antithetic sampling) sequences:

```
Closed-form vanilla call option price =
MC vanilla call price with Park-Miller uniforms =
MC vanilla call price with Park-Miller uniforms and antithetics =
QMC vanilla call price with Sobol sequence =
```

You should adapt *one* the following Sobol sequence generators:

- *J. Burkhardt*: `http://people.scs.fsu.edu/~burkardt/cpp_src/sobol/sobol.html`.
- *S. Joe and F. Y. Kuo*: `http://web.maths.unsw.edu.au/~fkuo/sobol/`.

Burkhardt's code generates Sobol sequences up to $d = 1,111$, while the Joe and Kuo's code generates Sobol sequences up to $d = 21,000$; you are asked to use $d = 12$, so either code will be adequate.

When testing and debugging your code, you may wish to compare the output of your chosen Sobol sequence generator output with that of a simpler Sobol sequence generator in a low dimension:

- Brandimarte: `GetSobol.m` from `staff.polito.it/paolo.brandimarte/`
- *BRODA*: `www.broda.co.uk/software.html`
- *GNU Scientific Library*: sample main test program `gsl_qrng_sobol_main.cpp` and `Makefile` showing how to compile with g++ on Linux.

- *Numerical Recipes*: `sobseq.h` (uses `nr3.h`, dimension $d \leq 6$), with sample main test program `xsobseq.cpp`. This code is supplied for testing and debugging only — do NOT use it in your submitted code.
- *Quantlib*: `sobolrsg.hpp`, `sobolrsg.cpp`.

GSL is recommended for use on Linux.

Be careful that while sequences of d calls to the Park-Miller algorithm produce sequences of vector samples from the uniform distribution on $[0,1]^d$, one must use the d-dimensional Sobol sequence to produce points in $[0,1]^d$; one *cannot* replace the Park-Miller algorithm directly by d calls to the 1-dimensional Sobol generator when $d > 1$ and your program must be coded accordingly.

(c) *Program submission. Unmodified* source and header files from Joshi or *Numerical Recipes* (`sobseq.h` and `nr3.h`) can be assumed available via include statements such as

```
#include <Arrays.h>
```

and should *not* need to be submitted or included inline. The lines `double tmp; cin >> tmp;` should be removed from Joshi's sample main program.

(d) **Benchmarking.** Benchmark your results using Excel spreadsheets of Haug, Back, or Rouah-Vainberg, the MATLAB functions of Brandimarte or Mathworks' toolboxes.

(e) **Report.** Write a report (Word or LaTeX) which includes

- A short explanation of the algorithms and their implementation and an analysis of your results, including a comparison of your results with those obtain from an Excel-VBA spreadsheet or MATLAB algorithm.
- Answers to the questions below:
  - Explain why we must use the inverse transform (`Random2.h` and `Random2.cpp`) rather than the Marsaglia-Bray-Box-Müller algorithm (`Random1.h` and `Random1.cpp`) for generating normals in conjunction with the Sobol sequence.
  - Use MATLAB, Excel, GnuPlot, Mathematica, Maple, or any other graphing program to plot $1,024$ pairs of samples from the uniform distribution over the unit square, $[0,1] \times [0,1]$, using the Park-Miller generator, and then $1,024$ pairs from the 2-dimensional Sobol sequence.

3. Practice Exercises and Review Questions

Problem 3.1. Consult the given references and explain how to generate Halton, Faure, and Sobol sequences.

Problem 3.2. Explain how to derive the closed-form formula for the discretely sampled geometric Asian option. Why does the method not apply to discretely sampled arithmetic Asian options?

Problem 3.3. Brandimarte provides MATLAB code to generate Halton and Sobol sequences (section 4.6), as well code to price an arithmetic Asian option (section 8.4) using quasi-Monte Carlo with the Halton sequence and Brownian bridge path generation (section 8.1.3). Examine Brandimarte's algorithm and compare results with your C++ code.

Problem 3.4. Given a bivariate normal random variable, (X, Y), with mean μ and covariance matrix Σ, derive the formula for the distribution of X conditional on $Y = y$. Generalize to the case where (X, Y) is a multivariate normal random variable and X and Y are vector-valued normal random variables and derive the formula in Equation (2.25) in Glasserman.

Problem 3.5. Describe the multi-dimensional Brownian bridge and modify your Brownian bridge class to generate paths of multi-dimensional Brownian motion.

Problem 3.6. If $W(t)$ is Brownian motion on $(\Omega, \mathbb{P}, \mathscr{F})$ and $0 \leq s_1 < \cdots < s_k < \infty$ with $s \in (s_i, s_{i+1})$, explain why the conditional distribution of $W(s)$ given $W(s_1), \ldots, W(s_k)$ is equal to the conditional distribution of $W(s)$ given $W(s_i)$ and $W(s_{i+1})$.

Problem 3.7. Review the three main methods of generating sample paths of Brownian motion (see Theorem 3.3.2 in Shreve). What are the advantages of the Brownian bridge over other methods?

Problem 3.8. Continuous and discretely sampled Asian options can be priced using Excel-VBA spreadsheets provided by Haug (section 4.20) and Back (sections 8.9 and 8.10). Compare their Excel-VBA results with those obtained using your C++ code.

Problem 3.9. Compare the theoretical error estimates and error decay rates for Monte carlo, quasi-Monte Carlo, and traditional numerical integration methods. What are the most useful ranges of application for each method?

4. Supplementary Notes

4.1. **Low-discrepancy sequences.** The definitions of discrepancy and star discrepancy and the definitions of low-discrepancy sequences of points in $[0,1]^d$ and algorithms for their construction (Halton, Faure, and Sobol) for $d \geq 1$ are explained in sections 5.1 and 5.2 of Glasserman; because their construction is quite technical, students are not expected to be familiar with the detailed algorithms (except for students studying such sequences as part of their final project) or definitions of discrepancy, except for the concept that low-discrepancy sequences provide an efficient method of "filling" the hypercube $[0,1]^d$ with a smaller number points for quasi-Monte Carlo integration than might be required by traditional numerical integration for the same accuracy.

4.2. **Quasi-Monte Carlo.** In quasi-Monte Carlo, the problem of estimating the expectation of a random variable is reformulated as a deterministic integral of a function f over $[0,1]^d$ and sequences of samples from the uniform distribution on $[0,1]^d$ forming paths with d time points for Monte Carlo simulation are replaced by d-dimensional low-discrepancy sequences for quasi-Monte Carlo integration. One can obtain an acceleration convergence from $O(1/\sqrt{n})$ in MC to $O(1/n^{1-\varepsilon})$, for any $\varepsilon > 0$, at least for moderate dimensions. When the dimension becomes very large, QMC may become less effective. In contrast to MC, statistical confidence intervals or error bounds are not meaningful for QMC (although they for "randomized QMC", as described in section 5.4 in Glasserman) and the Koksma-Hlawka error bound (in terms of the Hardy-Krause variation of f and the star-discrepancy of the sequence) are impractical to compute and often replaced by convergence tables, though these can be misleading. QMC and its application to finance are explained in more detail in Glasserman, section 5.1 and 5.5.

The advantages of MC or QMC over deterministic numerical integration (whose grid points would qualify as low-discrepancy sequences) include more flexibility in the definition of the "function" f and speed of convergence in higher dimensions; for example, the product trapezoidal rule converges as $O(1/n^{2/d})$ and so MC would generally converge faster than the product trapezoidal rule when $d \geq 5$ while QMC would generally converge faster when $d \geq 3$.

4.3. **Brownian bridge path generation.** The Brownian bridge and its application to the construction of sequences of m-dimensional Brownian motion sampled at n time points are discussed in section 3.1.1 (for $m = 1$) and 3.1.2 (for $m > 1$) in Glasserman; the method requires the conditional distribution of the multi-variate normal random vector in section 2.3.1 of Glasserman (p. 65).