Stochastic Simulation
M. Giles

Practical 2

The purpose of this practical is to learn how to perform Sobol QMC computations, investigate approximations of SDEs, and learn a bit more about Multilevel Monte Carlo (MLMC).

1. Look at the Matlab code `lec5c.m` which generated the comparison of Monte Carlo, Latin Hypercube and Sobol QMC performance in Lecture 5 for the basket option.

   Convert the codes to C++, python or R (depending which group you are in).

   For C++, I suggest using the code from Frances Kuo and Stephen Joe which is available here. You will need to implement digital shift scrambling yourselves – talk to me if you have questions about how to do this.

   For python, use scipy.stats.qmc.Sobol which supports Owen scrambling.

   For R, you can use either the SobolSequence or the RandToolbox CRAN packages; I think the first of these supports digital shift scrambling, while the second supports Owen scrambling.

   If you are interested to learn more about quasi-Monte Carlo methods, an excellent reference is the 2013 Acta Numerica review article High-dimensional integration: The quasi-Monte Carlo way by Josef Dick, Frances Kuo and Ian Sloan.

2. Look at the Matlab codes `lec7_weak.m` and `lec7_strong.m` which produced the plots in Lecture 7, and make sure that you understand what they are doing – ask if anything is unclear. Note that $g(e+\Delta e) \approx g(e) + \Delta e\, g'(e)$, so that if e is an estimate for $\mathbb{E}[f]$ with confidence interval $\pm 3\sigma/\sqrt{N}$ then $g(e)$ is an estimate for $g(\mathbb{E}[f])$ with confidence interval $\pm 3(\sigma/\sqrt{N})\, g'(e)$; this is used in `lec7_strong.m` to obtain a confidence interval for $\sqrt{\mathbb{E}[(\Delta S)^2]}$.

   Convert the codes to C++, python or R.

   For the C++ code, I suggest you create an output file with the results data which you can then read into Matlab or python to do the plotting.

3. Modify `lec7_strong` to simulate the mean-reverting Ornstein-Uhlenbeck process

$$\mathrm{d}S = \kappa\,(\theta - S)\,\mathrm{d}t + \sigma\,\mathrm{d}W$$

   with $S(0) = 100, \theta = 110, \kappa = 2, \sigma = 0.5$ over the time interval $[0, 1]$. There is no exact solution in this case so just plot the comparison between the h and $2h$ solutions.

   What is the order of strong convergence?

4. Modify it again for the Heston stochastic volatility model which is a coupled pair of SDEs:

$$\begin{aligned} \mathrm{d}S &= r\,S\,\mathrm{d}t + \sqrt{|v|}\,S\,\mathrm{d}W^{(1)} \\ \mathrm{d}v &= \kappa\,(\theta - v)\,\mathrm{d}t + \xi\,\sqrt{|v|}\,\mathrm{d}W^{(2)} \end{aligned}$$

with $S(0) = 100, v(0) = 0.25, \theta = 0.25, \kappa = 2, \xi = 0.5$ over the time interval $[0, 1]$.

The two driving Brownian motions are correlated so that

$$\mathbb{E}[\mathrm{d}W^{(1)}\mathrm{d}W^{(2)}] = -0.1\,\mathrm{d}t$$

so the correlation matrix is

$$\Sigma = \begin{pmatrix} 1 & -0.1 \\ -0.1 & 1 \end{pmatrix}.$$

Again there is no exact solution in this case so just plot the comparison between the h and $2h$ solutions.

What is the order of strong convergence?

5. Download and read my original MLMC paper Multilevel Monte Carlo path simulation

From my MLMC software webpage:

- python groups: follow the link there to a bitbucket repository – the "opre" example has code to more-or-less replicate the results in my original paper (the original calculations were done in MATLAB using a different random number generator). If there are any problems in using the bitbucket repository then I think this zip file has the same code.
- C++ groups: the "mcqmc06" C++ code is for a different set of experiments in a second paper: Improved multilevel Monte Carlo convergence using the Milstein scheme
- R groups: the link there takes you to the MLMC CRAN page; I think it also has the "opre" example which is more-or-less the same as in the original paper
- all groups: run the codes, see the results you get, and then read through the codes in detail

In future assignments you will be asked to create new Multilevel Monte Carlo (MLMC) applications, so the point of this assignment is to understand the software structure – the routines like "mlmc" and "mlmc_test" are generic, the same for every application, and what the user has to write is the low-level "routine_l" code which computes the output correction on a particular MLMC level for the particular application of interest. For more details, see section 3 in a more recent Acta Numerica paper, Multilevel Monte Carlo methods.

For further reading, if you are interested, see my MLMC community webpage and my MLMC research webpage.