Monte Carlo Methods for Uncertainty Quantification: Practical 2

This practical is about the use of MLMC (multilevel Monte Carlo) for uncertainty quantification in two settings.

1. Download from the course webpage the Matlab routines `mlmc_test.m`, `mlmc.m`, `gbm.m`, `ellip.m`.

   - `mlmc_test.m` performs a number of tests for an MLMC application.
   - `mlmc.m` does the main MLMC computation, working out the optimal number of samples to use on each level of approximation.
   - `gbm.m` is an application for a financial option based on an underyling stock represented by a Geometric Brownian Motion model.
   - `ellip.m` is a very simple 1D elliptic solver with random forcing.

2. Start with the `gbm.m` application. Look carefully at the routine `gbm_l` which computes N_ℓ fine path samples S^f and coarse path samples S^c, and the corresponding payoff functions P_ℓ^f and $P_{\ell-1}^c$, for a given level ℓ.

   Run the code and see the results it produces. Look at the code and see how the fine and coarse paths are computed; check that this matches the explanation given in the lectures.

   Note that in `gbm_l` the sample paths are computed in groups of 10,000. This is a trick to minimise the overheads in MATLAB. If programmed in C / C++ / FORTRAN you would typically do one path at a time.

   The Euler discretisation is not very accurate. Modify the code to instead use the Milstein approximation:

   $$S_{n+1} = S_n + r\, S_n\, \Delta t + \sigma S_n \Delta W_n + \tfrac{1}{2}\sigma^2 S_n (\Delta W_n^2 - \Delta t)$$

   This gives first order strong convergence, so you should see the multilevel variance decay more quickly with level.

3. `ellip.m` solves the 1D elliptic PDE:

   $$u''(x) = 100\, Z \sin(\pi x), \quad 0 < x < 1$$

   where Z is a standard Normal random variable (zero mean and unit variance), and the boundary conditions are $u(0) = u(1) = 0$.

The output of interest is taken to be

$$P = \int_0^1 u^2(x)\ dx.$$

A simple finite difference approximation is used for the PDE, and the integral is approximated by trapezoidal integration.

Since the coefficients of the tri-diagonal matrix do not vary, the matrix is precomputed. This then allows us to compute samples 100 at a time to minimise the MATLAB overhead.

(a) Modify the code so that it is solving

$$(a\,u')' = 100, \quad 0 < x < 1$$

with $a(x) = \exp(-Z \sin(\pi x))$, where Z is again a standard Normal random variable, and the boundary conditions are still $u(0) = u(1) = 0$.

Note that in this case you will need to generate a separate matrix for each random sample, and so you will need to process all of the samples one by one.

(b) Modify the code so that the boundary conditions are $u(0) = 0$, $u(1) = Z_2$, where Z_2 is a second independent standard Normal random variable.

(c) If you want, you can also experiment with different output functions.