Stochastic Simulation
M. Giles

Practical 1

The purpose of this practical is just to get used to random number generation within your preferred language (python, Matlab, C/C++, R), and try out a few basic things from lectures 1-4.

Please work on this as a group of 3 or 4, and hand in an archive file (.tar or .zip) with i) your code(s), ii) your results as a PDF file with comments on anything you think is interesting.

1. (a) Generate 10^6 uniformly distributed variables on the unit interval $[0, 1]$, and check that they have the expected mean and variance.

   Repeat for 10^6 unit Normal random variables.

   Notes: in python, you can use `rand, randn` from `numpy.random`; in Matlab you can use `rand, randn`; for C++ see the example code I've put in the course materials.

   (b) Given a covariance matrix

$$\Sigma = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$$

   perform a Cholesky factorisation to obtain a lower-triangular matrix L such that

$$\Sigma = L\,L^T$$

   Use this matrix L to convert 2×10^6 independent unit Normals into 10^6 pairs of Normals with the desired covariance. Check that they have the expected mean and covariance.

   Notes: in python there is a function `cholesky` in `numpy.linalg`; in Matlab there is a function `chol`; for C++ work out the Cholesky factorisation mathematically.

   (c) Repeat the previous item using the PCA factorisation of Σ.

   Notes: in python there is a function `eig` in `numpy.linalg`; in Matlab there is a function `eig`; for C++ work out the eigenvalues and eigenvectors mathematically.

   (d) Repeat to see how many pairs you can generated in 1 minute.

   (It will be interesting to compare the python and C++ results on this performance test; we can make the C++ go even faster when we switch to using multi-threading and vectorisation with Intel's MKL.)

2. Let U be uniformly distributed on $[0, 1]$. You are to use Monte Carlo simulation to estimate the value of

$$\overline{f} = \mathbb{E}[f(U)] = \int_0^1 f(U)\ \mathrm{d}U$$

where

$$f(x) = x\ \cos \pi x.$$

(a) Calculate analytically the exact value for $\overline{f}$ and

$$\sigma^2 = \mathbb{E}[(f(U) - \overline{f})^2] = \int_0^1 (f(U) - \overline{f})^2\ \mathrm{d}U$$

(b) Write a program which to compute

$$Y_m = N^{-1} \sum_{n=1}^{N} f(U^{(m,n)})$$

for 1000 different sets of 1000 independent random variables $U^{(m,n)}$.

(c) Sort the Y_m into ascending order, and then plot $C_m = (m - 1/2)/1000$ versus Y_m – this is the numerical cumulative distribution function.

Superimpose on the same plot the cumulative distribution function you would expect from the Central Limit Theorem, and comment on your results.

You may like to experiment by trying larger or smaller sets of points to improve your understanding of the asymptotic behaviour described by the CLT.

For those doing experiments in C++, I suggest you do the plotting in python.

(d) Modify your code to use a single set of 10^6 random numbers, and plot

$$Y_N = N^{-1} \sum_{n=1}^{N} f(U^{(n)})$$

versus N for $N = 10^3 - 10^6$. This should demonstrate the convergence to the true value predicted by the Strong Law of Large Numbers.

For each N, also compute an unbiased estimate for the variance σ^2 and hence add to the plot upper and lower confidence bounds based on 3 standard deviations of the variation in the mean.

Add a line corresponding to the true value. Does this lie inside the bounds?

3. Repeat Question 2 for a European call option in which the final value of the underlying is

$$S(T) = S(0)\ \exp\left((r-\tfrac{1}{2}\sigma^2)T + \sigma W(T)\right)$$

where

$$W(T) = \sqrt{T}\ X = \sqrt{T}\ \Phi^{-1}(U)$$

with X being a unit Normal, or U a uniform $(0,1)$ random variable.

The payoff function is

$$f(S) = \exp(-rT)\ (S(T) - K)^+$$

and the constants are $r = 0.05, \sigma = 0.2, S(0) = 100, K = 100$.

The analytic value is given by the routine `european_call` available from my webpage; read its header to see how to call it.

There is no need to compute the analytic variance in part a); just use the unbiased estimator when plotting the CLT prediction in part c).

4. For the case of Geometric Brownian Motion and a European call option, with parameters, $r=0.05,\ \sigma=0.2,\ T=1,\ S(0)=100,\ K=100$, investigate the following forms of variance reduction:

   (a) First, try antithetic variables using $\frac{1}{2}\left(f(W) + f(-W)\right)$ where W is the value of the underlying Brownian motion at maturity.

   What is the estimated correlation between $f(W)$ and $f(-W)$? How much variance reduction does this give?

   (b) Second, try using $\exp(-rT)\,S(T)$ as a control variate, noting that its expected value is $S(0)$.

   Again, how much variance reduction does this give?

5. For the case of a digital put option,

$$P = \exp(-r\,T)\ H(K - S_T)$$

where $H(x)$ is the Heaviside step function, and S_T is described by Geometric Brownian Motion

$$\log S_T = \log S_0 + (r - \tfrac{1}{2}\sigma^2)T + \sigma\sqrt{T}\ Y$$

with parameters $r = 0.05,\ \sigma = 0.2,\ T = 1,\ S(0) = 100,\ K = 50$, investigate the use of importance sampling:

   (a) First, estimate the value without importance sampling.

   How many samples are needed to obtain a value which is correct to within 10%? (i.e. the 3 standard deviation confidence limit corresponds to $\pm$ 10%).

   (b) Second, try using importance sampling, adjusting the drift (i.e. changing the $(r - \frac{1}{2}\sigma^2)T$ term to a different constant) so that half of the samples are below the strike K, and the other half are above.

   Now how many samples are required to get the value correct to within 10%?

6. (a) For the same application as question 4, use the finite difference "bumping" method to compute delta and vega, the sensitivities to changes in the initial price and the volatility.

   Check your results are correct by comparing to the analytic values given by `european_call`.

   Experiment with and without using the same random numbers for the two sets of samples to see the effect on the variance.

   Also experiment with the size of the "bump" to see the effect on the accuracy.

   (b) Re-do the calculation using the IPA ("pathwise" sensitivity) method.