Stochastic Simulation: Lecture 1

Prof. Mike Giles

Oxford University Mathematical Institute

Monte Carlo methods

Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ where

- Ω is the sample space of all possible outcomes
- $\mathcal{F}$ is the σ-algebra of events (subsets of Ω)
- $\mathbb{P}$ is the associated probabilities for these events

then to estimate $\mathbb{E}[P(\omega)]$, where P is some scalar quantity of interest, the simplest Monte Carlo estimate is

$$N^{-1} \sum_{n=1}^{N} P(\omega^{(n)})$$

where the $\omega^{(n)} \in \Omega$ are i.i.d. samples from the appropriate distribution.

Monte Carlo methods

Note that lots of different quantities can be expressed as an expectation:

- mean: $\mathbb{E}[P(\omega)]$
- mean-variance combination:
  $\mathbb{E}\left[\frac{1}{2}(P(\omega_1)+P(\omega_2))+\frac{1}{2}\gamma(P(\omega_1)-P(\omega_2))^2\right] = \mathbb{E}[P]+\gamma\mathbb{V}[P]$
- probabilities: $\mathbb{E}[\mathbf{1}_{Q>\beta}] = \mathbb{P}[Q>\beta]$

Random Number Generation

Monte Carlo simulation usually starts with random number generation, which often is split into 2 stages:

- generation of independent uniform $(0,1)$ random variables
- conversion into independent random variables from some other distribution (e.g. Normal)

Generating “good” uniform random variables is technically complex, so **never** write your own generator, **always** use a well validated generator from a reputable source

- Matlab
- NAG
- Intel MKL
- AMD ACML
- **not** MS Excel, C `rand` function or Numerical Recipes

Uniform Random Variables

Pseudo-random number generators use a deterministic (i.e. repeatable) algorithm to generate a sequence of (apparently) random numbers on $(0, 1)$ interval.

What defines a good generator?

- a long period – how long it takes before the sequence repeats itself

  2^{32} is not enough – need at least 2^{40}
- various statistical tests to measure "randomness"

  well validated software will have gone through these checks

Uniform Random Variables

Practical considerations:

- computational cost – RNG cost can be as large as rest of Monte Carlo simulation
- trivially-parallel Monte Carlo simulation on a compute cluster requires the ability to "skip-ahead" to an arbitrary starting point in the sequence

  first computer gets first 10^6 numbers
  second computer gets second 10^6 numbers, etc

Uniform Random Variables

“Multiplicative congruential algorithms” based on

$$n_i = (a \times n_{i-1}) \mod m$$

- choice of integers a and m is crucial
- (0,1) random number given by n_i/m
- typical period is 2^{57}, a bit smaller than m
- can skip-ahead 2^k places at low cost by repeatedly squaring a, mod m

Uniform Random Variables

For more details see

- Intel MKL information

  `software.intel.com/en-us/`

  `mkl-developer-reference-c-random-number-generators`

- NAG library information

  `www.nag.co.uk/numeric/CL/nagdoc_cl08/pdf/G05/g05_conts.pdf`

- Matlab information

  `www.mathworks.com/moler/random.pdf`

- Wikipedia information

  `en.wikipedia.org/wiki/Random_number_generation`

  `en.wikipedia.org/wiki/List_of_random_number_generators`

  `en.wikipedia.org/wiki/Mersenne_Twister`

Normal Random Variables

Applications often require Normal random variables, $N(\mu, \sigma^2)$, with mean μ and variance σ^2.

An $N(0,1)$ Normal random variable Z with mean 0, variance 1 has a probability density function (pdf)

$$\phi(z) = \frac{1}{\sqrt{2\pi}} \exp(-\tfrac{1}{2} z^2),$$

and cumulative distribution function (CDF)

$$\Phi(z) = \mathbb{P}[Z < z] = \int_{-\infty}^{z} \phi(s) \, \mathrm{d}s.$$

Normal Random Variables

To generate $N(0,1)$ Normal random variables, we start with a sequence of uniform random variables on $(0,1)$.

There are then 4 main ways of converting them into $N(0,1)$ Normal variables:

- Box-Muller method
- Marsaglia's polar method (ignored – doesn't vectorise well)
- Marsaglia's ziggurat method (ignored – doesn't vectorise well)
- inverse CDF transformation

Normal Random Variables

The Box-Muller method takes y_1, y_2, two independent uniformly distributed random variables on $(0,1)$ and defines

$$\begin{aligned} x_1 &= \sqrt{-2\log(y_1)}\ \cos(2\pi y_2) \\ x_2 &= \sqrt{-2\log(y_1)}\ \sin(2\pi y_2) \end{aligned}$$

It can be proved that x_1 and x_2 are $N(0,1)$ random variables, and independent.

A log, cos and sin operation per 2 Normals makes this a slightly expensive method.

Normal Random Variables

The inverse CDF transformation method takes y, uniformly distributed on $(0, 1)$, and defines

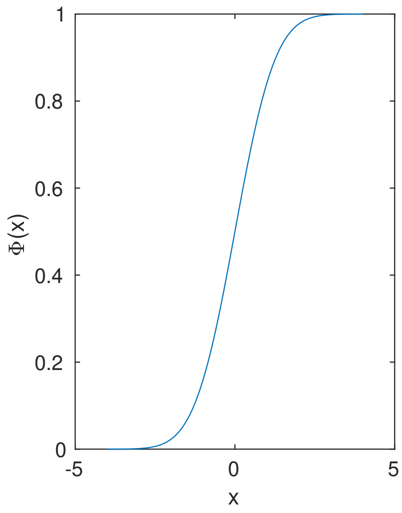
$$x = \Phi^{-1}(y),$$

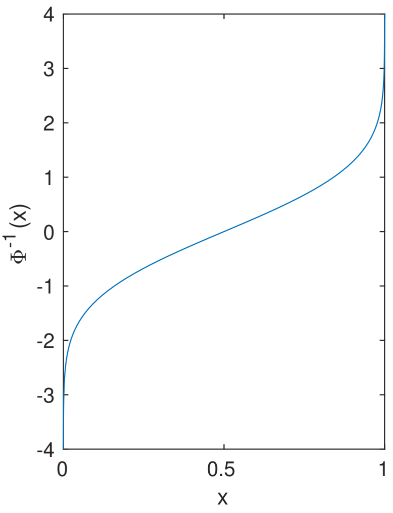
where $\Phi(x)$ is the Normal CDF defined earlier.

$\Phi^{-1}(y)$ is approximated in software in a very similar way to the implementation of $\cos, \sin, \log$, so this is just as accurate as the other methods.

It is also a more flexible approach because we'll need $\Phi^{-1}(y)$ later for stratified sampling and quasi-Monte Carlo methods.

Normal Random Variables





Normal Random Variables

Some useful weblinks:

- `home.online.no/ pjacklam/notes/invnorm/`
  code for Φ^{-1} function in many different languages
- `lib.stat.cmu.edu/apstat/241/`
  single and double precision code in FORTRAN
  (coming soon in next version of NAG libraries)
- `en.wikipedia.org/wiki/Normal_distribution`
  Wikipedia definition of Φ matches mine
- `mathworld.wolfram.com/NormalDistribution.html`
  `mathworld.wolfram.com/DistributionFunction.html`
  Good Mathworld items, but their definition of Φ is slightly different; they call the cumulative distribution function $D(x)$.

Normal Random Variables

The Normal CDF $\Phi(x)$ is related to the error function $\mathrm{erf}(x)$:

$$\Phi(x) = \tfrac{1}{2} + \tfrac{1}{2}\mathrm{erf}(x/\sqrt{2}) \quad \Longrightarrow \quad \Phi^{-1}(y) = \sqrt{2}\,\mathrm{erf}^{-1}(2y-1)$$

so this is the function I often use in Matlab code:

```
function x = ncfinv(y)
x = sqrt(2)*erfinv(2*y-1);
```

However, the MATLAB Statistics toolbox also has a function

```
norminv(p)
norminv(p,mu,sigma).
```

Correlated Normal Random Variables

Sometimes we need a vector of Normally distributed variables with a prescribed covariance matrix.

Suppose x is a vector of independent $N(0,1)$ variables, and define a new vector $y = L\,x$.

Each element of y is Normally distributed, $\mathbb{E}[y] = L\,\mathbb{E}[x] = 0$, and

$$\mathbb{E}[y\,y^T] = \mathbb{E}[L\,x\,x^T\,L^T] = L\,\mathbb{E}[x\,x^T]\,L^T = L\,L^T.$$

since $\mathbb{E}[x\,x^T] = I$ because

- elements of x are independent $\Longrightarrow \mathbb{E}[x_i\,x_j] = 0$ for $i \neq j$
- elements of x have unit variance $\Longrightarrow \mathbb{E}[x_i^2] = 1$

Correlated Normal Random Variables

To get $\mathbb{E}[y\,y^T] = \Sigma$, we need to find L such that $L\,L^T = \Sigma$

L is not uniquely defined. Simplest choice is to use a Cholesky factorization in which L is lower-triangular. but alternatively, if Σ has eigenvalues $\lambda_i \geq 0$, and orthonormal eigenvectors u_i, so that

$$\Sigma\,u_i = \lambda_i\,u_i, \quad \Longrightarrow \quad \Sigma\,U = U\,\Lambda$$

then

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

where

$$L = U\,\Lambda^{1/2}.$$

This is the PCA decomposition; it is no better than the Cholesky decomposition for standard Monte Carlo simulation, but is often better for quasi-Monte Carlo methods.

Expectation and Integration

If x is a random variable uniformly distributed on $[0,1]$ then the expectation of a function $f(x)$ is equal to its integral:

$$\mathbb{E}[f(x)] = \int_0^1 f(x)\,\mathrm{d}x.$$

The generalisation to a d-dimensional “cube” $I^d = [0,1]^d$, is

$$\mathbb{E}[f(x)] = \int_{I^d} f(x)\,\mathrm{d}x.$$

Thus finding expectations is directly connected to the problem of numerical quadrature (integration), often in very large dimensions.

Expectation and Integration

Suppose we have a sequence x_n of independent samples from the uniform distribution.

An approximation to the expectation/integral is given by

$$\overline{f}_N = N^{-1} \sum_{n=1}^{N} f(x_n).$$

Two key features:

- Unbiased: $\mathbb{E}\left[\overline{f}_N\right] = \mathbb{E}[f]$
- Convergent: $\lim_{N \to \infty} \overline{f}_N = \mathbb{E}[f]$

CLT

This MC estimate is unbiased, meaning that the average error is zero

$$\mathbb{E}[\varepsilon_N] = 0$$

where $\varepsilon_N = \overline{f}_N - \mathbb{E}[f]$.

In addition, the Central Limit Theorem proves that for large N the error is asymptotically Normally distributed

$$\varepsilon_N(f) \sim \sigma\, N^{-1/2}\, Z$$

with Z a $N(0,1)$ random variable and σ^2 the variance of f:

$$\sigma^2 = \mathbb{V}[f] \equiv \mathbb{E}\left[(f - \mathbb{E}[f])^2\right].$$

CLT

This means that

$$\mathbb{P}\left[\left|N^{1/2}\sigma^{-1}\varepsilon_N\right| < s\right] \;\approx\; 1 - 2\,\Phi(-s),$$

where $\Phi(s)$ is the Normal CDF (cumulative distribution function).

Typically we use $s = 3$, corresponding to a 3-standard deviation confidence interval, with $1 - 2\,\Phi(-s) \approx 0.997$.

Hence, with probability 99.7%, we have

$$\left|N^{1/2}\sigma^{-1}\varepsilon_N\right| < 3 \quad\Longrightarrow\quad |\varepsilon_N| < 3\,\sigma\,N^{-1/2}$$

This bounds the accuracy, but we need an estimate for σ.

Empirical Variance

Given N samples, the empirical variance is

$$\widetilde{\sigma}^2 = N^{-1} \sum_{n=1}^{N} \left(f^{(n)} - \overline{f}_N \right)^2 = \overline{f^2}_N - (\overline{f}_N)^2$$

where

$$\overline{f}_N = N^{-1} \sum_{n=1}^{N} f^{(n)}, \qquad \overline{f^2}_N = N^{-1} \sum_{n=1}^{N} \left(f^{(n)} \right)^2$$

$\widetilde{\sigma}^2$ is a slightly biased estimator for σ^2 – an unbiased estimator is

$$\widehat{\sigma}^2 = \frac{N}{N-1}\,\widetilde{\sigma}^2 = \frac{N}{N-1}\left(\overline{f^2}_N - (\overline{f}_N)^2 \right)$$

Expectation and Integration

How does Monte Carlo integration compare to grid based methods for d-dimensional integration?

MC error is proportional to $N^{-1/2}$ independent of the dimension.

If the integrand is sufficiently smooth, trapezoidal integration with $M = N^{1/d}$ points in each direction has

$$\text{Error} \ \propto \ M^{-2} \ = \ N^{-2/d}$$

This scales better than MC for $d < 4$, but worse for $d > 4$. i.e. MC is better at handling high dimensional problems.

Application

As a simple example, the Black-Scholes model uses a geometric Brownian motion model for a single asset:

$$S_T = S_0 \ \exp\left((r - \tfrac{1}{2}\sigma^2) T + \sigma \, W_T \right)$$

where W_T is the value of the Brownian motion at time T, and has a Normal distribution with mean 0, variance T;

From this we will calculate an expected value

$$V = \mathbb{E}\left[f(S_T) \right].$$

Application

We can put

$$W_T = \sqrt{T}\ Y = \sqrt{T}\ \Phi^{-1}(U)$$

where Y is a $N(0,1)$ random variable, and U is uniformly distributed on $[0,1]$.

Thus

$$V = \mathbb{E}\left[f(S_T)\right] = \int_0^1 f(S_T)\ \mathrm{d}U,$$

with

$$\begin{aligned} S_T &= S_0\ \exp\left((r - \tfrac{1}{2}\sigma^2)T + \sigma\sqrt{T}\ Y\right) \\ &= S_0\ \exp\left((r - \tfrac{1}{2}\sigma^2)T + \sigma\sqrt{T}\,\Phi^{-1}(U)\right) \end{aligned}$$

Application

For the European call option,

$$f(S) = \exp(-rT)\ \max(S-K, 0)$$

while for the European put option

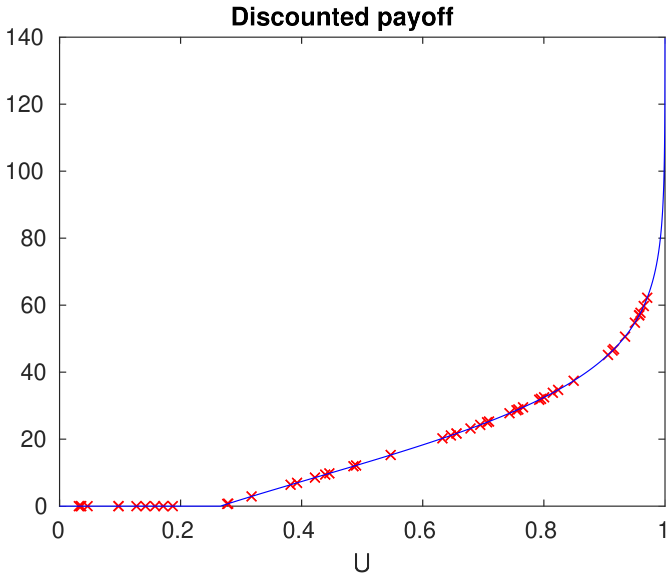
$$f(S) = \exp(-rT)\ \max(K-S, 0)$$

where K is the strike price.

For numerical experiments we will consider a European call with $r = 0.05, \quad \sigma = 0.2, \quad T = 1, \quad S_0 = 110, \quad K = 100.$

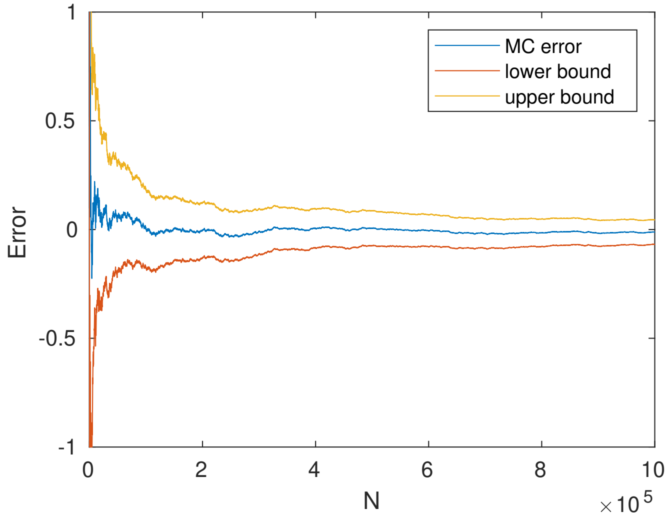
The analytic value is known for comparison.

Application



Application

MC calculation with up to 10^6 paths; true value $= 17.663$



Applications

The upper and lower bounds are given by

$$\mathrm{Mean} \pm \frac{3\,\widetilde{\sigma}}{\sqrt{N}},$$

so more than a 99.7% probability that the true value lies within these bounds.

Applications

MATLAB code:

```
r=0.05;  sig=0.2;  T=1;  S0=110;  K=100;
N = 1:1000000;
U = rand(1,max(N));    % uniform random variable
Y = norminv(U);        % inverts Normal cum. fn.
S = S0*exp((r-sig^2/2)*T + sig*sqrt(T)*Y);
F = exp(-r*T)*max(0,S-K);

sum1 = cumsum(F);      % cumulative summation of
sum2 = cumsum(F.^2);   % payoff and its square
val  = sum1./N;
rms  = sqrt(sum2./N - val.^2);
```

Applications

```
err = european_call(r,sig,T,S0,K,'value') - val;

plot(N,err,                      ...
     N,err-3*rms./sqrt(N),  ...
     N,err+3*rms./sqrt(N))
axis([0 length(N) -1 1])
xlabel('N'); ylabel('Error')
legend('MC error','lower bound','upper bound')
```

Final Words

- Monte Carlo quadrature is straightforward and robust
- Confidence bounds can be obtained as part of the calculation
- Can calculate the number of samples N needed for chosen accuracy
- Much more efficient than grid-based methods for high dimensions
- Accuracy $= O(N^{-1/2})$, CPU time $= O(N)$

  $\Longrightarrow \quad \text{accuracy} = O(\text{CPU time}^{-1/2})$

  $\Longrightarrow \quad \text{CPU time} = O(\text{accuracy}^{-2})$