Numerical Methods II

Prof. Mike Giles

mike.giles@maths.ox.ac.uk

Oxford University Mathematical Institute

Monte Carlo methods

In computational finance for option pricing there are two main approaches:

- Monte Carlo methods for estimating expected values of financial payoff functions based on underlying assets.

  This term, we consider payoffs which depend on the terminal value of one or more underlying assets.

  In the simplest case, we have

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

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

Monte Carlo methods

- Numerical approximation of the PDE which describes the evolution of the expected value.

$$u(s,t) = \mathbb{E}\left[f(S_T) \mid S_t = s\right]$$

This is usually less costly than MC when there are very few underlying assets (≤ 3), but much more expensive when there are many.

Random Number Generation

Monte Carlo simulation starts with random number generation, which often is split into 3 stages:

- generation of independent uniform $(0, 1)$ random variables
- conversion into independent Normal $N(0, 1)$ random variables
- conversion into correlated Normal $N(0, 1)$ random variables

This lecture will cover:

- what you need to know as a user
- background information to make you better informed

Uniform Random Variables

- Generating “good” uniform random variables is technically complex
- **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
- What you need to know is what to look for in a good generator
- Useful background knowledge is how they work

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

Mersenne twister is very popular in finance:

- developed in 1997 so still quite new
- huge period of $2^{19937}-1$; I think this is the main reason it's popular for Monte Carlo applications
- I've heard conflicting comments on its statistical properties

Uniform Random Variables

For more details see

- Intel MKL information

  `www.intel.com/cd/software/products/asmo-na/eng/266864.htm`

- 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

In computational finance we work extensively with 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 and 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
- Marsaglia's ziggurat method
- 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

Marsaglia's polar method is as follows:

- Take y_1, y_2 from uniform distribution on $(-1, 1)$
- Accept if $r^2 = y_1^2 + y_2^2 < 1$, otherwise get new y_1, y_2
- Define

$$x_1 = \sqrt{-2\log(r^2)/r^2}\ y_1$$
$$x_2 = \sqrt{-2\log(r^2)/r^2}\ y_2$$

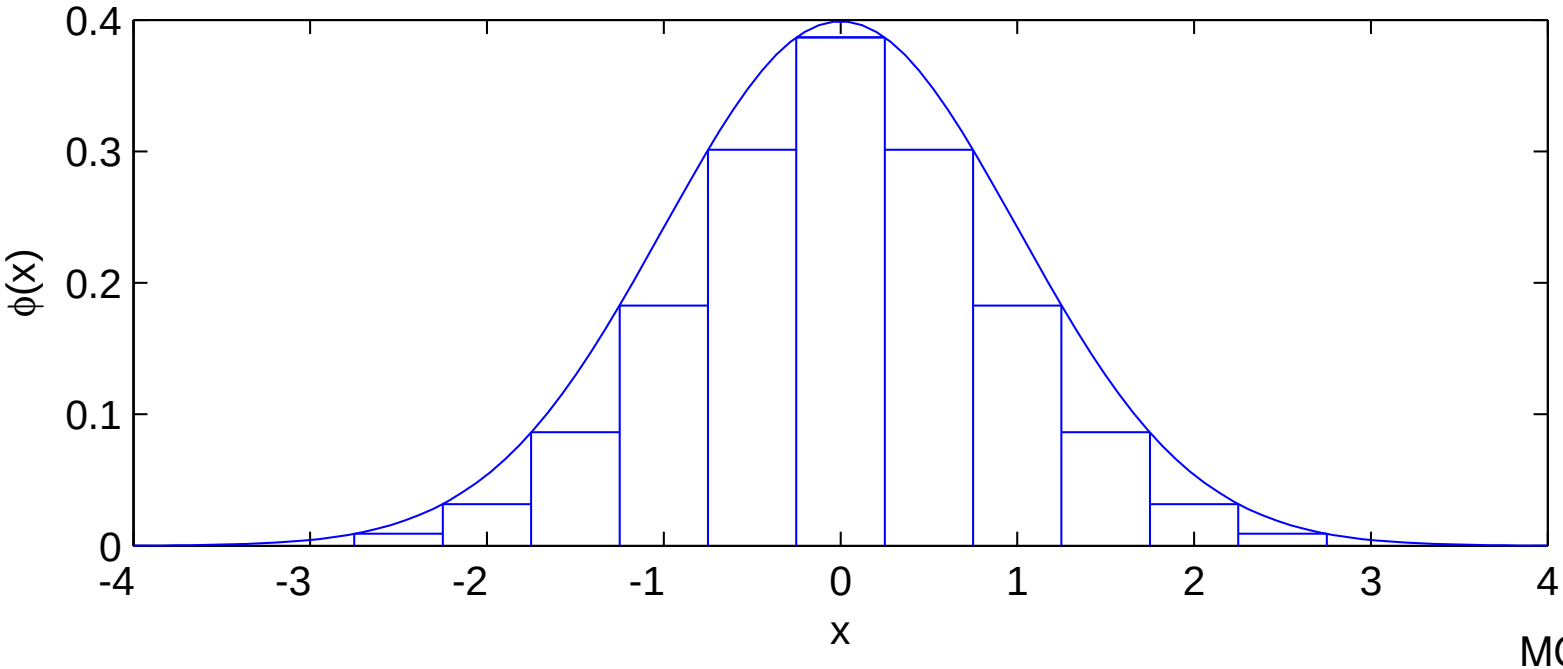
Again it can be proved that x_1 and x_2 are independent $N(0, 1)$ random variables.

Despite approximately 20% rejections, it is slightly more efficient because of not needing $\cos, \sin$ operations. However, rejection of some of the uniforms spoils the skip-ahead capability.

Normal Random Variables

Marsaglia's ziggurat method is the fastest, but also the hardest to explain. The details aren't really important, so I'll just give an outline.

The unit area under the standard Normal distribution in broken up into a number of rectangles (accounting for over 98% of the area) and other bits.



Normal Random Variables

Given a uniformly distributed $(0, 1)$ random input:

- a lookup table is used to determine if it corresponds to one of the rectangles
- if so, then the corresponding output is uniformly distributed within the rectangle and so can be computed very easily
- if not, the calculation is much more complex, but this only happens 2% of the time
- Matlab uses this approach in `randn`
  `www.mathworks.com/moler/random.pdf`

Normal Random Variables

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

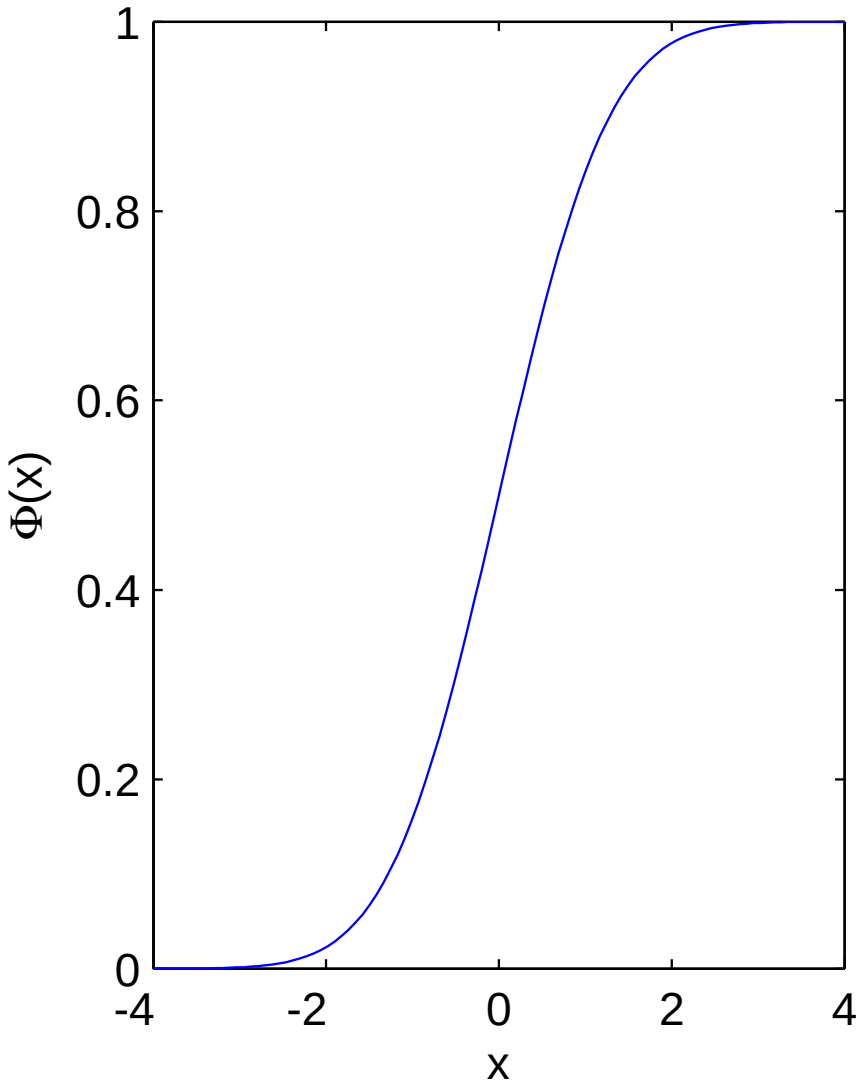
$$x = \Phi^{-1}(y),$$

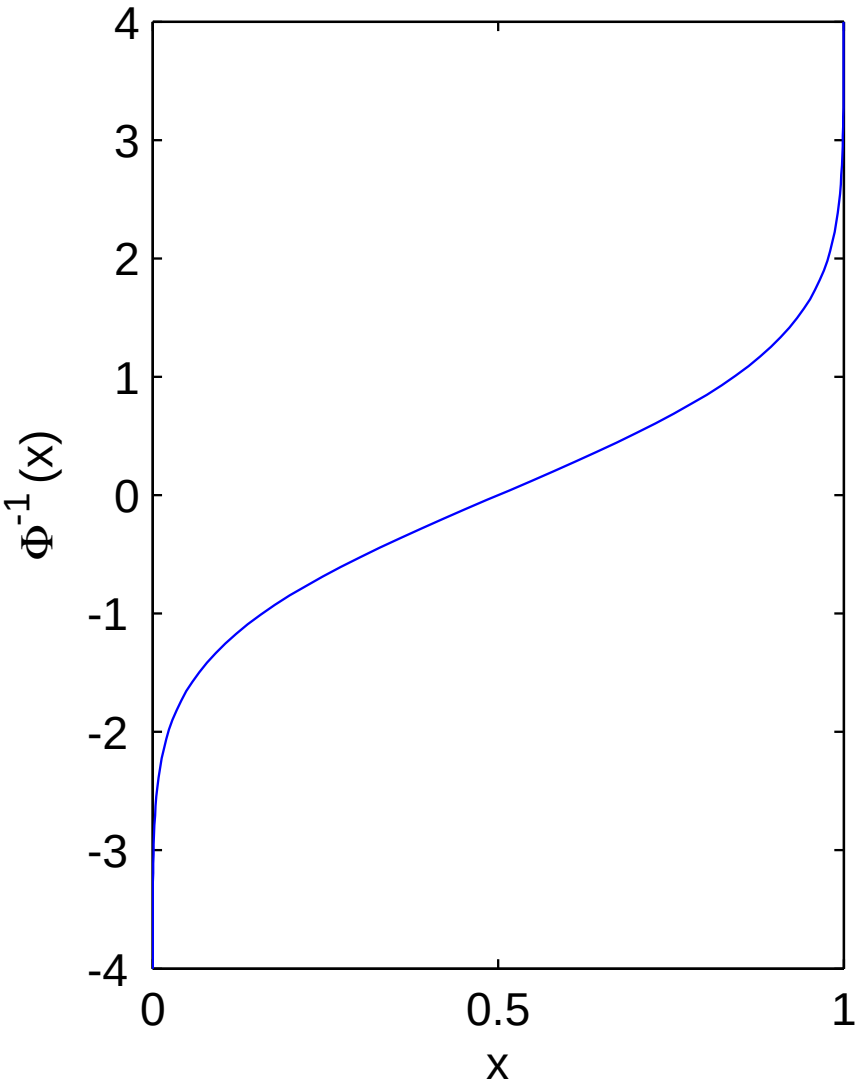
where $\Phi(x)$ is the Normal CDF defined earlier.

$\Phi^{-1}(y)$ 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

The final step is to generate 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, with a positive diagonal.

Correlated Normal Random Variables

Pseudo-code for Cholesky factorization $LL^T = \Sigma$:

```
for i from 1 to N
   for j from 1 to i
      for k from 1 to j-1
         Σ_ij := Σ_ij - L_ik L_jk
      end

      if j=i
         L_ii := sqrt(Σ_ii)
      else
         L_ij := Σ_ij / L_jj
      endif
   end
end
```

Correlated Normal Random Variables

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 stratified sampling and quasi-Monte Carlo methods.

Final advice

- **always** use mathematical libraries as much as possible
- usually they will give you uncorrelated Normals, and you have to convert these into correlated Normals
- later with stratified sampling and quasi-Monte Carlo methods, we will use the inverse cumulative Normal distribution to convert (quasi-)uniforms into (quasi-)Normals