

# Monte Carlo Methods for Uncertainty Quantification

Mike Giles

Mathematical Institute, University of Oxford

ERCOFTAC course on Mathematical Methods and Tools  
in Uncertainty Management and Quantification

October 25, 2013

# Lecture outline

## Lecture 1: Introduction and Monte Carlo basics

- some model applications
- random number generation
- Monte Carlo estimation
- Central Limit Theorem and confidence interval

## Lecture 2: Variance reduction

- basic manipulations
- control variate
- importance sampling
- stratified sampling
- Latin Hypercube
- randomised quasi-Monte Carlo

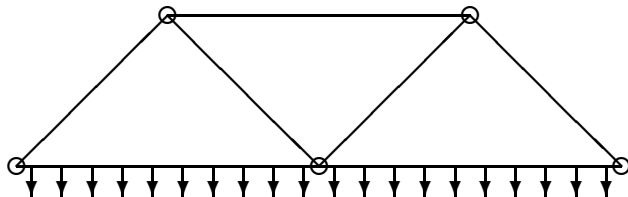
# Lecture outline

## Lecture 3: multilevel Monte Carlo and applications

- weak and strong convergence
- mean square error decomposition
- multilevel Monte Carlo
- PDEs with uncertainty

## Application 1

Consider a bridge with 7 elements and pinned joints:



Design will compute a force balance, work out compression / extension of each element, and therefore determine the natural length to be cast.

However, the manufactured elements will vary from design in both length and extensibility – 14 uncertain inputs.

If two supporting joints have fixed position, then analysis has 6 unknowns (coordinates of free joints) and 6 equations (force balance at free joints).

# Application 1

Given manufacturing data on the variability of the natural length and extensibility, what might we want to know?

- RMS deviation of joint position from design
- RMS deviation of forces from design
- probability of maximum compression / extension force being outside some specified range

Note: if we turn this into a full finite element analysis, then the computational cost becomes much larger.

## Application 2

Consider a square trampoline, with vertical position given by

$$T \left( \frac{\partial^2 Z}{\partial x^2} + \frac{\partial^2 Z}{\partial y^2} \right) = L(x, y), \quad 0 < x < 1, \quad 0 < y < 1$$

where  $T$  is the tension and  $L(x, y)$  is the applied load.

Here the uncertainty could be in the boundary conditions:

- simplest case would be uncertainty in the 4 corner values of  $Z(x, y)$  with straight line interpolation along each edge
- a more complicated case might add a Fourier decomposition of the perturbation from the straight line interpolation

$$Z(x, 0) = (1-x) Z_{0,0} + x Z_{1,0} + \sum_{n=1}^{\infty} a_n \sin(n\pi x)$$

Could also have uncertainty in the tension and the loading.

## Application 2

Again there are various outputs we might be interested in:

- average values for  $\min Z(x, y)$  and  $\max Z(x, y)$
- RMS variation in these due to uncertainty

Note: biggest displacements likely to occur in the middle, not significantly affected by high order Fourier perturbations on the boundary.

## Application 3

In modelling groundwater flow in nuclear waste repositories, or oil flow in oil reservoirs, we use Darcy's Law:

$$\nabla \cdot (\kappa \nabla p) = 0$$

where  $p$  is the pressure and  $\kappa$  is the permeability of the rock.

The uncertainty here is in  $\kappa(x)$  – typically might know the values at a few bore holes, but there is a large amount of uncertainty elsewhere.

We do know that if two points  $x_1, x_2$  are close, then  $\kappa(x_1) \approx \kappa(x_2)$ , but if they are far apart then they can be quite different. Hence, often model  $\log \kappa$  as having a Normal distribution, with a spatial covariance of the form

$$\text{cov}(\log \kappa(x_1), \log \kappa(x_2)) = \sigma^2 \exp(-\|x_1 - x_2\|/\lambda)$$

where  $\lambda$  is the correlation length.



## Application 3

What does this application require?

- generation of samples of the stochastic field  $\log \kappa(x)$  with correct distribution
- computation of  $p(x)$  by solving Darcy PDE
- evaluation of outputs of interest (e.g. water or oil flux across some boundary)
- Monte Carlo simulation to obtain average value, RMS variation, probability of exceeding some threshold, etc.

# Random Number Generation

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

- generation of independent uniform  $(0, 1)$  random variables
- conversion into random variables with a particular distribution (e.g. Normal)

**Very important:** never write your own generator, always use a well validated generator from a reputable source

- Matlab
- NAG
- Intel MKL
- AMD ACML

# 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
- 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

For information see

- Intel MKL information

[www.intel.com/cd/software/products/asm-na/eng/266864.htm](http://www.intel.com/cd/software/products/asm-na/eng/266864.htm)

- NAG library information

[www.nag.co.uk/numeric/CL/nagdoc\\_c108/pdf/G05/g05\\_conts.pdf](http://www.nag.co.uk/numeric/CL/nagdoc_c108/pdf/G05/g05_conts.pdf)

- Matlab information

[www.mathworks.com/moler/random.pdf](http://www.mathworks.com/moler/random.pdf)

- Wikipedia information

[en.wikipedia.org/wiki/Random\\_number\\_generation](http://en.wikipedia.org/wiki/Random_number_generation)

[en.wikipedia.org/wiki/List\\_of\\_random\\_number\\_generators](http://en.wikipedia.org/wiki/List_of_random_number_generators)

[en.wikipedia.org/wiki/Mersenne\\_Twister](http://en.wikipedia.org/wiki/Mersenne_Twister)

# Normal Random Variables

$N(0, 1)$  Normal random variables (mean 0, variance 1) have the probability distribution

$$p(x) = \phi(x) \equiv \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$$

The Box-Muller method takes two independent uniform  $(0, 1)$  random numbers  $y_1, y_2$ , 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:

$$p_{\text{joint}}(x_1, x_2) = p(x_1) p(x_2)$$

## Inverse CDF

A more flexible alternative uses the cumulative distribution function  $CDF(x)$  for a random variable  $X$ , defined as

$$CDF(x) = \mathbb{P}(X < x)$$

If  $Y$  is a uniform  $(0, 1)$  random variable, then can define  $X$  by

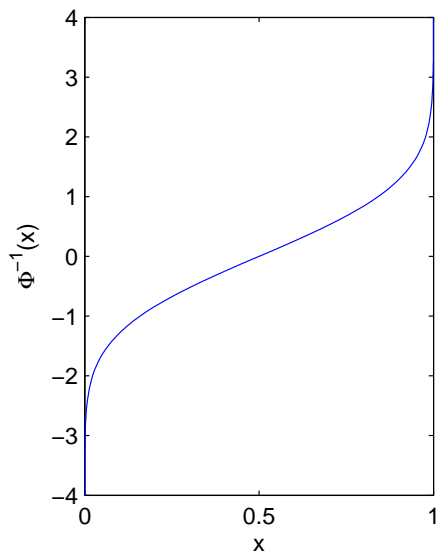
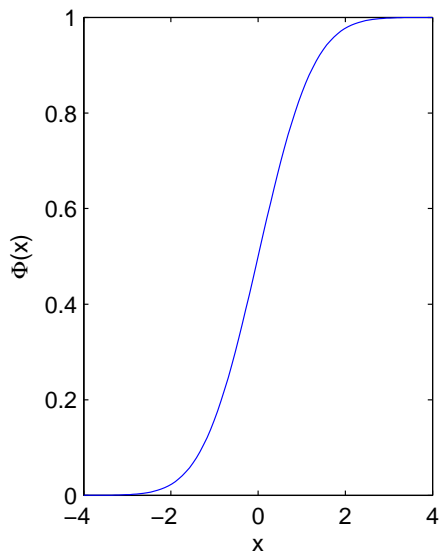
$$X = CDF^{-1}(Y).$$

For  $N(0, 1)$  Normal random variables,

$$CDF(x) = \Phi(x) \equiv \int_{-\infty}^x \phi(s) ds = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}s^2\right) ds$$

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

# Normal Random Variables



# Normal Random Variables

Some useful weblinks:

- [en.wikipedia.org/wiki/Normal\\_distribution](http://en.wikipedia.org/wiki/Normal_distribution)  
Wikipedia definition of  $\Phi$  matches mine
- [mathworld.wolfram.com/NormalDistribution.html](http://mathworld.wolfram.com/NormalDistribution.html)  
[mathworld.wolfram.com/DistributionFunction.html](http://mathworld.wolfram.com/DistributionFunction.html)  
Good Mathworld items, but their definition of  $\Phi$  is slightly different; they call the cumulative distribution function  $D(x)$ .
- [lib.stat.cmu.edu/apstat/241/](http://lib.stat.cmu.edu/apstat/241/)  
single and double precision code in FORTRAN
- <http://people.maths.ox.ac.uk/gilesm/erfinv/>  
My GPU CUDA code for inverse error function `erfinv`



# Normal Random Variables

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

$$\Phi(x) = \frac{1}{2} + \frac{1}{2} \text{erf}(x/\sqrt{2}) \implies \Phi^{-1}(y) = \sqrt{2} \text{erf}^{-1}(2y-1)$$

This is the function I use in Matlab:

```
% x = ncfinv(y)
%
% inverse Normal CDF

function x = ncfinv(y)

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

## Correlated Normal Random Variables

We often need a vector  $y$  of Normally distributed variables with a prescribed covariance matrix, so that  $\mathbb{E}[y y^T] = \Sigma$ .

Suppose  $x$  is a vector of independent  $N(0, 1)$  variables, and define  $y = Lx$ . 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  $\implies \mathbb{E}[x_i x_j] = 0$  for  $i \neq j$
- elements of  $x$  have unit variance  $\implies \mathbb{E}[x_i^2] = 1$

Hence choose  $L$  so that  $LL^T = \Sigma$ . One choice is a Cholesky factorisation in which  $L$  is lower-triangular. Another is PCA in which  $L = U\Lambda^{1/2}$  with  $U$  the matrix of eigenvectors of  $\Sigma$ , and  $\Lambda$  the diagonal eigenvalue matrix.

# 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:

$$\bar{f} = \mathbb{E}[f(X)] = I[f] = \int_0^1 f(x) dx.$$

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

$$\bar{f} = \mathbb{E}[f(X)] = I[f] = \int_{I^d} f(x) dx.$$

Thus the problem of 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

$$I_N[f] = N^{-1} \sum_{n=1}^N f(x^{(n)}).$$

Two key features:

- Unbiased:  $\mathbb{E} [I_N[f]] = I[f]$
- Convergent:  $\lim_{N \rightarrow \infty} I_N[f] = I[f]$

# Central Limit Theorem

In general, define

- error  $\varepsilon_N(f) = I[f] - I_N[f]$
- bias =  $\mathbb{E}[\varepsilon_N(f)]$
- RMSE, “root-mean-square-error” =  $\sqrt{\mathbb{E}[(\varepsilon_N(f))^2]}$

The Central Limit Theorem proves (roughly speaking) that for large  $N$

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

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

$$\sigma^2 = \mathbb{E}[(f - \bar{f})^2] = \int_{I^d} (f(x) - \bar{f})^2 dx.$$

# Central Limit Theorem

More precisely, provided  $\sigma$  is finite, then as  $N \rightarrow \infty$ ,

$$\text{CDF}(N^{1/2}\sigma^{-1}\varepsilon_N) \rightarrow \text{CDF}(Z)$$

so that

$$\mathbb{P}\left[N^{1/2}\sigma^{-1}\varepsilon_N < s\right] \rightarrow \mathbb{P}[Z < s] = \Phi(s)$$

and

$$\mathbb{P}\left[\left|N^{1/2}\sigma^{-1}\varepsilon_N\right| > s\right] \rightarrow \mathbb{P}[|Z| > s] = 2\Phi(-s)$$

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

# Central Limit Theorem

Given  $N$  samples, the empirical variance is

$$\tilde{\sigma}^2 = N^{-1} \sum_{n=1}^N \left( f(x^{(n)}) - I_N \right)^2 = I_N^{(2)} - (I_N)^2$$

where

$$I_N = N^{-1} \sum_{n=1}^N f(x^{(n)}), \quad I_N^{(2)} = N^{-1} \sum_{n=1}^N \left( f(x^{(n)}) \right)^2$$

$\tilde{\sigma}^2$  is a slightly biased estimator for  $\sigma^2$ ; an unbiased estimator is

$$\hat{\sigma}^2 = (N-1)^{-1} \sum_{n=1}^N \left( f(x^{(n)}) - I_N \right)^2 = \frac{N}{N-1} \left( I_N^{(2)} - (I_N)^2 \right)$$

# Central Limit Theorem

How many samples do we need for an accuracy of  $\bar{\varepsilon}$  with probability  $c$ ?

Since

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

define  $s$  so that  $1 - 2 \Phi(-s) = c$

$c$	0.683	0.9545	0.9973	0.99994
$s$	1.0	2.0	3.0	4.0

Then  $|\varepsilon| < N^{-1/2} \sigma s$  with probability  $c$ , so to get  $|\varepsilon| < \bar{\varepsilon}$  we can put

$$N^{-1/2} \hat{\sigma} s(c) = \bar{\varepsilon} \quad \implies \quad N = \left( \frac{\hat{\sigma} s(c)}{\bar{\varepsilon}} \right)^2.$$

Note: twice as much accuracy requires 4 times as many samples.



# 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.

## Other outputs

Using Monte Carlo we can compute more than just simple averages.

First of all, can compute quantities like the standard deviation:

$$\sigma_f^2 = \mathbb{E} [(f - \mathbb{E}[f])^2]$$

and other higher moments.

## Other outputs

We can also approximate the cumulative distribution function

$$C(s) = \mathbb{P}[f(X) < s] = \mathbb{E}[\mathbf{1}_{f(X) < s}]$$

or the probability density function,  $P(s) = \frac{dC}{ds}$ , in various ways:

- Maximum Entropy reconstruction – uses  $\mathbb{E}[f^m]$  for  $m = 1, 2, \dots, M$  to construct PDF approximation  $P(s) \approx \exp(p(s))$ , where  $p(s)$  is a polynomial and  $P(s)$  has the same moments
- Alternatively, can evaluate  $C(s_j)$  for a set of values  $s_j$ , and then interpolate to approximate the full  $C(s)$

## Summary so far

- 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 =  $O(N)$ 
  - $\implies$  accuracy =  $O(\text{CPU time}^{-1/2})$
  - $\implies$  CPU time =  $O(\text{accuracy}^{-2})$
- the key now is to reduce number of samples required by reducing the variance