Monte Carlo Methods for Uncertainty Quantification

Mike Giles

Mathematical Institute, University of Oxford

Contemporary Numerical Techniques

Lecture outline

Lecture 1: Monte Carlo basics

- motivating applications
- basics of mean and variance
- random number generation
- Monte Carlo estimation
- Central Limit Theorem and confidence interval

Lecture 2: Variance reduction

- control variate
- Latin Hypercube
- randomised quasi-Monte Carlo

Lecture outline

Lecture 3: financial applications

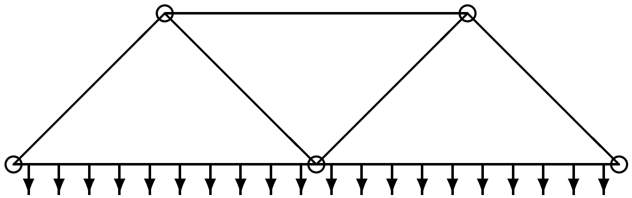
- financial models
- approximating SDEs
- weak and strong convergence
- mean square error decomposition
- multilevel Monte Carlo

Lecture 4: PDE applications

- PDEs with uncertainty
- examples
- multilevel Monte Carlo

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

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

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 computational finance, the behaviour of assets (e.g.stocks) is modelled by stochastic differential equations such as:

$$\mathrm{d}S = r\, S\, \mathrm{d}t + \sigma\, S\, \mathrm{d}W$$

where $\mathrm{d}W$ is the increment of a Brownian motion, which is Normally distributed with zero mean and variance $\mathrm{d}t$.

The stochastic term $\sigma\, S\, \mathrm{d}W$ models the uncertainty in the day-to-day evolution of the asset price, due to random events.

We will not cover the theory of Ito calculus which is necessary to work with SDEs, but it can be proved that

$$\mathrm{d} \log S = (r - \tfrac{1}{2}\sigma^2)\, \mathrm{d}t + \sigma\, \mathrm{d}W$$

and hence

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

Application 3

Later, we will consider a basket call option based on 5 assets, each with

$$S_i(T) = S_i(0)\ \exp\left((r - \tfrac{1}{2}\sigma_i^2)T + \sigma_i W_i(T)\right)$$

and with the option value being

$$f = \exp(-rT)\ \max\left(0, \tfrac{1}{5}\sum_i S_i(T) - K\right)$$

What we want to estimate is the expected value of this option.

Objective

In general, we

- start with a random sample ω
- usually compute some intermediate quantity U
- then evaluate a scalar output $f(U)$

$$\omega \ \rightarrow \ U \ \rightarrow \ f(U)$$

The objective is then to compute the **expected** (or average) value

$$\mathbb{E}[f(U)]$$

Basics

In some cases, the random inputs are discrete: X has value x_i with probability p_i, and then

$$\mathbb{E}[f(X)] = \sum_i f(x_i)\, p_i$$

In other cases, the random inputs are continuous random variables: scalar X has probability density $p(x)$ if

$$\mathbb{P}(X \in (x, x{+}\mathrm{d}x)) = p(x)\,\mathrm{d}x + o(\mathrm{d}x)$$

Then

$$\mathbb{E}[f(X)] = \int f(x)\, p(x)\, \mathrm{d}x$$

In either case, if a, b are random variables, and λ, μ are constants, then

$$\begin{aligned} \mathbb{E}[a+\mu] &= \mathbb{E}[a] + \mu \\ \mathbb{E}[\lambda\, a] &= \lambda\, \mathbb{E}[a] \\ \mathbb{E}[a+b] &= \mathbb{E}[a] + \mathbb{E}[b] \end{aligned}$$

Basics

The **variance** is defined as

$$\begin{aligned}\mathbb{V}[a] &\equiv \mathbb{E}\left[(a-\mathbb{E}[a])^2\right] \\ &= \mathbb{E}\left[a^2 - 2a\,\mathbb{E}[a] + (\mathbb{E}[a])^2\right] \\ &= \mathbb{E}\left[a^2\right] - (\mathbb{E}[a])^2\end{aligned}$$

It then follows that

$$\begin{aligned}\mathbb{V}[a+\mu] &= \mathbb{V}[a] \\ \mathbb{V}[\lambda\, a] &= \lambda^2\,\mathbb{V}[a] \\ \mathbb{V}[a+b] &= \mathbb{V}[a] + 2\,\mathrm{Cov}[a,b] + \mathbb{V}[b]\end{aligned}$$

where
$$\mathrm{Cov}[a,b] \equiv \mathbb{E}\left[(a-\mathbb{E}[a])\,(b-\mathbb{E}[b])\right]$$

Basics

X_1 and X_2 are independent continuous random variables if

$$p_{\rm joint}(x_1,x_2) = p_1(x_1)\ p_2(x_2)$$

We then get

$$\begin{aligned}
\mathbb{E}[f_1(X_1)\ f_2(X_2)] &= \iint f_1(x_1)\ f_2(x_2)\ p_{\rm joint}(x_1,x_2)\ {\rm d}x_1\ {\rm d}x_2 \\
&= \iint f_1(x_1)\ f_2(x_2)\ p_1(x_1)\ p_2(x_2)\ {\rm d}x_1\ {\rm d}x_2 \\
&= \left(\int f_1(x_1)\ p_1(x_1)\ {\rm d}x_1\right)\left(\int f_2(x_2)\ p_2(x_2)\ {\rm d}x_2\right) \\
&= \mathbb{E}[f_1(X_1)]\ \ \mathbb{E}[f_2(X_2)]
\end{aligned}$$

Hence, if a, b are independent, ${\rm Cov}[a,b]\!=\!0 \Longrightarrow \mathbb{V}[a\!+\!b] = \mathbb{V}[a] + \mathbb{V}[b]$
More generally, the variance of the sum of independent r.v.'s is the sum of their variances.

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
- Intel MKL

Uniform Random Variables

Pseudo-random 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

For information see

- Intel MKL information

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

  `www.mathworks.com/moler/random.pdf`
- Wikipedia information

  `en.wikipedia.org/wiki/Random_number_generation`

Normal Random Variables

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

$$p(x) \;=\; \phi(x) \;\equiv\; \frac{1}{\sqrt{2\pi}}\,\exp(-\tfrac{1}{2}x^2)$$

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:

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

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

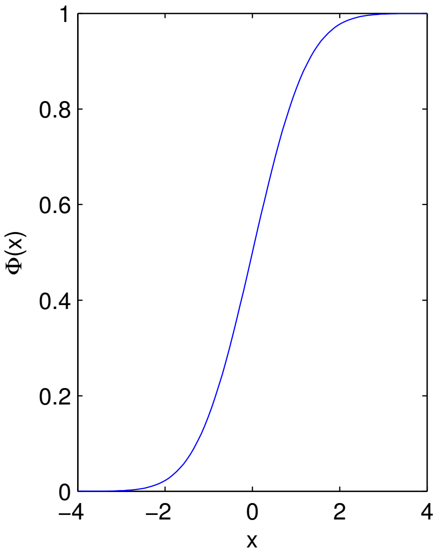
$$X = CDF^{-1}(Y).$$

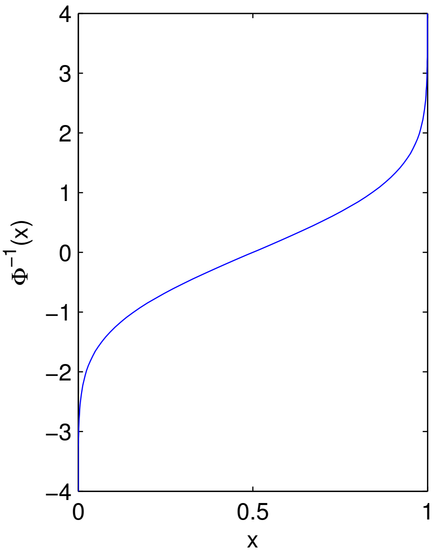
For $N(0,1)$ Normal random variables,

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

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

Normal Random Variables





MATLAB

- `rand(n,m)` generates a matrix of independent r.v.'s each uniformly distributed on unit interval $(0,1)$
- `randn(n,m)` generates a matrix of independent r.v.'s each Normally distributed with zero mean and unit variance
- `clear all` at the beginning of your MATLAB code will reset everything, including the random number generators, so every time you run the program you get the **same** "random" numbers
- `norminv(u)` computes $\Phi^{-1}(u)$
- `normcdf(x)` computes $\Phi(x)$
- `normpdf(x)` computes $\phi(x)$

Expectation and Integration

If X is a random variable uniformly distributed on $[0,1]$ then its probability density function is

$$p(x) = \begin{cases} 1, & 0 < x < 1 \\ 0, & \text{otherwise} \end{cases}$$

and therefore

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

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

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

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

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

Note that this is an unbiased estimator, since for each n,

$$\mathbb{E}[f(X_n)] = \mathbb{E}[f(X)] = I[f]$$

and therefore

$$\mathbb{E}\left[I_N[f] \right] = I[f]$$

Central Limit Theorem

In general, define

- error $\quad \varepsilon_N(f) = I[f] - I_N[f]$
- 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 σ^2 the variance of f:

$$\sigma^2 = \mathbb{V}[f(X)] = \int_{I^d} (f(x) - I[f])^2 \;\mathrm{d}x.$$

provided σ^2 is finite.

Central Limit Theorem

More precisely, provided σ is finite, then as $N \longrightarrow \infty$,

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

so that

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

and

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

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

Estimated Variance

Given N samples, the empirical variance is

$$\widetilde{\sigma}^2 = N^{-1} \sum_{n=1}^{N} (f(x_n) - I_N)^2 = I_N^{(2)} - (I_N)^2$$

where

$$I_N = N^{-1} \sum_{n=1}^{N} f(x_n), \qquad I_N^{(2)} = N^{-1} \sum_{n=1}^{N} (f(x_n))^2$$

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

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

Confidence Interval

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

Since

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

define s so that $\quad 1-2\,\Phi(-s) = c \quad\Longleftrightarrow\quad s = -\Phi^{-1}((1-c)/2)$

c	0.683	0.9545	0.9973	0.99994
s	1.0	2.0	3.0	4.0

$|\varepsilon| < N^{-1/2}\,\sigma\,s$ with probability c – this is the confidence interval.

To ensure $|\varepsilon| < \overline{\varepsilon}$ with probability c we can put

$$N^{-1/2}\,\widehat{\sigma}\,s(c) = \overline{\varepsilon} \quad\Longrightarrow\quad N = \left(\frac{\widehat{\sigma}\,s(c)}{\overline{\varepsilon}}\right)^2.$$

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

Summary so far

- Monte Carlo estimation / 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
- accuracy $= O(N^{-1/2})$, CPU time $= O(N)$

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

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