The Monte Carlo method

Felipe Uribe

Computational Engineering
School of Engineering Sciences
Lappeenranta-Lahti University of Technology (LUT)

Special Course on Inverse Problems
Lappeenranta, FI — January-February, 2024

Why Monte Carlo?

- Oftentimes, UQ objectives are given in terms of expectations or variances of random variables of interest.
- Simulation of random variables is a fundamental process in the application of UQ methods.
- When random variables are unknown (e.g., inverse problems), sampling with advanced Monte Carlo-based methods is necessary.
- It is robust to the type and dimension of the problem. Moreover, it is simple to apply.

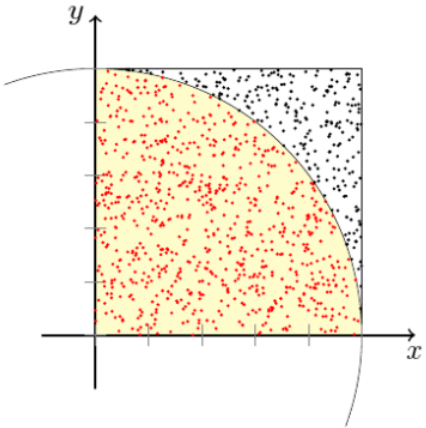

Figure: Monte Carlo method to estimate π.

Monte Carlo method: basic timeline

- Enrico Fermi (1930) - studying neutron diffusion.
- Stanislaw Ulam (1940) - Markov chain Monte Carlo method while he was working on nuclear weapons (Los Alamos).
- John von Neumann (1946) - neutron diffusion in the core of a nuclear weapon (Los Alamos).
- Nicholas Metropolis (1946) - suggested using the name Monte Carlo (Manhattan Project).

Figure: JvN.

This Lecture...

- The lecture is based on multiple references. However, we mostly follow Chapters 2, 3 and 4 of the book by Art Owen[1], which is freely available online.

[1] A. B. Owen. *Monte Carlo theory, methods and examples*. `artowen.su.domains/mc/`, 2018.

Monte Carlo simulation

- "*Being secret, the work of von Neumann and Ulam required a code name. A colleague of them, Nicholas Metropolis, suggested using the name* ***Monte Carlo****, which refers to the Monte Carlo casino in Monaco.*"

- The *Monte Carlo* (MC) method is a simulation procedure in which the quantity we want to compute, typically an integral of a given response variable $\boldsymbol{Y} = g(\boldsymbol{X})$, is expressed as an expected value over the distribution of $\boldsymbol{X}$.

- MC relies on repeated random sampling and statistical analysis to obtain numerical results.

- Simple MC = Crude MC = Standard MC = MC integration.

Monte Carlo simulation

- Two fundamental questions arise when using MC simulation:
  - (i) How close is the MC estimator to the actual value of the expectation?
  - (ii) How do we sample from the distribution of $\boldsymbol{X}$?

- For the first question, one needs to justify and characterize properties of the MC estimator. This essentially requires application of the **laws of large numbers**.

- The second question is addressed by the theory of random variable generation. In turn these algorithms are based on pseudo-random number generation schemes (uniform draws between 0 and 1).

PART I: properties of the MC estimator

Monte Carlo method

- In probability, we are most of the time faced with the computation of expected values.

$$\boldsymbol{\mu} = \mathbb{E}_{\pi}[\boldsymbol{X}] = \int_{\Omega} \boldsymbol{X} \, \mathrm{d}\mathbb{P}_X = \int_{\mathbb{R}^d} \boldsymbol{x} \, \mathrm{d}F_X(\boldsymbol{x}) = \int_{\mathbb{R}^d} \boldsymbol{x} \, \pi_X(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \tag{1}$$

- Assuming that we can generate values $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ from the distribution of a random vector $\boldsymbol{X}$, the MC estimator of $\boldsymbol{\mu}$ is defined as

$$\boldsymbol{\mu} \approx \widehat{\boldsymbol{\mu}}_n := \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i, \qquad \{\boldsymbol{x}_i\}_{i=1}^{n} \overset{\text{iid}}{\sim} F_{\boldsymbol{X}} \tag{2}$$

- The primary justification of MC is through the **laws of large numbers** (LLN).

Justification of Monte Carlo: weak LLN I

- Let $X_1(\omega), X_2(\omega), \ldots, X_n(\omega)$ be sequence of independent and identically distributed (iid) RVs, each having same mean $\mu = \mathbb{E}[X_i(\omega)] < \infty$. We define the RV of partial averages:

$$M_n(\omega) = \frac{1}{n} \sum_{i=1}^{n} X_i(\omega); \tag{3}$$

note that $\widehat{\mu}_n$ is a realization of $M_n(\omega)$. Then, the weak LLN (aka Khinchin's law) says:

$$\lim_{n \to \infty} \mathbb{P}[|M_n - \mu| \geq \epsilon] = 0, \quad \text{holds for any } \epsilon > 0. \tag{4}$$

- The average converges in probability towards the expected value, we write $M_n \xrightarrow{\mathbb{P}} \mu$ when $n \to \infty$. Intuition: "*the chance of missing by more than ϵ goes to zero as the number of trials goes to infinity*".

Justification of Monte Carlo: weak LLN II

- We can prove the weak LLN using Chebyshev's inequality which requires further assuming that the RVs have finite variance $\mathbb{V}[X_i] = \sigma^2 < \infty$.

- In general, the weak LLN holds for any sequence having only finite mean (the proof requires more details).

Justification of Monte Carlo: strong LLN

- Let $X_1, X_2, \ldots, X_n$ be sequence of iid RVs, each having same mean $\mu = \mathbb{E}[X_i] < \infty$, and define the RV of partial averages M_n. Then, the strong LLN (aka Kolmogorov's law):

$$\mathbb{P}\Big[\lim_{n\to\infty} M_n = \mu\Big] = 1. \tag{5}$$

- The average converges almost surely towards the expected value, we write $M_n \overset{\text{a.s.}}{\to} \mu$ when $n \to \infty$. Intuition: "*the absolute error will eventually get below ϵ and stays there forever*".

- Proof relies on Markov's inequality (see,[2] p.62).

[2] J. S. Rosenthal. *A first look at rigorous probability theory*. 2nd ed. World Scientific Publishing Company, 2006.

Some remarks about the LLNs

- RVs that converge strongly (almost surely) are guaranteed to converge weakly (in probability).
- We assume that μ exists... *We will see cases where this fails.*
- While both LLNs tell us that averages will eventually produce an error as small as we like, neither tells us how large n has to be for this to happen...
- The situation improves markedly when X_i have in addition finite variance.

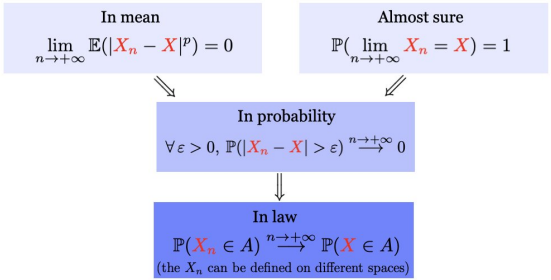

Figure: The LLNs are statements about convergence of RVs.

Accuracy of Monte Carlo I

- We suppose that the sequence of random variables has in addition finite variance $\mathbb{V}[X_i] = \sigma^2 < \infty$. The partial averages M_n represent a RV with its own mean and variance:

$$\mathbb{E}[M_n] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \mu \tag{6a}$$

$$\mathbb{V}[M_n] = \mathbb{E}\left[(M_n - \mu)^2\right] = \frac{\sigma^2}{n}. \tag{6b}$$

- Because $\mathbb{E}[M_n] = \mu$, we say that the MC estimator is **unbiased**.

Accuracy of Monte Carlo II

- A useful measure of the accuracy for the MC estimator is given by the **coefficient of variation** (cv):

$$\mathrm{cv} = \frac{\sqrt{\mathbb{V}[M_n]}}{\mathbb{E}[M_n]} = \frac{\sigma}{\sqrt{n}\mu}. \tag{7}$$

- We can also compute the **root mean squared error** (RMSE), to have a measure of the accuracy:

$$\mathrm{RMSE} = \sqrt{\mathbb{E}[(M_n - \mu)^2]} = \frac{\sigma}{\sqrt{n}}. \tag{8}$$

- Note that the cv and RMSE are of order $1/\sqrt{n}$.

Some remarks so far...

- The MC error has order $1/n^{1/2}$. This is very slow compared to other rates in many numerical methods. For instance, when $d = 1$, Simpson's method can integrate a function with an error of order $1/n^4$.
- While low accuracy cannot be considered a strength, it is not always a severe weakness. For example, MC scales better with increasing dimension compared to other quadrature methods: convergence rate of MC is independent of the dimension.
- The advantage of a MC approach is that we can put more real world complexity into our computations than we would be able to get with closed form estimates.

Accuracy of Monte Carlo: the CLT

- The central limit theorem (CLT) describes the limiting distribution of the averages M_n. It helps us to derive confidence intervals for our MC estimate.

- Let $X_1, X_2, \ldots, X_n$ be sequence of iid RVs, each having same mean $\mu = \mathbb{E}[X_i] < \infty$ and variance $\sigma^2 = \mathbb{V}[X_i] < \infty$. Then, the central limit theorem (CLT) holds:

$$\lim_{n\to\infty} \mathbb{P}\left[\sqrt{n}\frac{M_n - \mu}{\sigma} \leq x\right] = \Phi(x), \tag{9}$$

  where Φ denotes the standard Gaussian CDF.

- The CLT is a statement of convergence of CDFs (in distribution):

$$\sqrt{n}\frac{M_n - \mu}{\sigma} \xrightarrow{\mathsf{d}} \mathcal{N}(0,1) \qquad \text{or} \qquad M_n \xrightarrow{\mathsf{d}} \mathcal{N}(\mu, \sigma^2/n). \tag{10}$$

Accuracy of Monte Carlo: sketch of the CLT proof

- We know $\sum_{i=1}^{n} X_i$ has mean $n\mu$ and variance $n\sigma^2$. Consider the random variable:

$$\bar{S}_n = \frac{\sum_{i=1}^{n} X_i - n\mu}{\sqrt{n\sigma^2}} = \sum_{i=1}^{n} \frac{1}{\sqrt{n}} Y_i, \quad \text{with } Y_i = \frac{X_i - \mu}{\sigma}. \tag{11}$$

- The characteristic function of $\bar{S}_n$ is

$$\varphi_{\bar{S}_n}(t) = \mathbb{E}\left[\exp(jt\bar{S}_n)\right] = \varphi_{Y_1}(t/\sqrt{t}) \cdots \varphi_{Y_n}(t/\sqrt{t}) = \left(\varphi_{Y_1}(t/\sqrt{t})\right)^n. \tag{12}$$

- The characteristic function of Y_1 using Taylor expansion is

$$\varphi_{Y_1}(t/\sqrt{t}) = 1 - \frac{t^2}{2n} + \mathcal{O}(1/n). \tag{13}$$

- By the limit of the exponential function

$$\varphi_{\bar{S}_n}(t) = \left(1 - \frac{t^2}{2n} + \mathcal{O}(1/n)\right)^n \to \exp(-t^2/2), \quad n \to \infty. \tag{14}$$

Visualization of the CLT

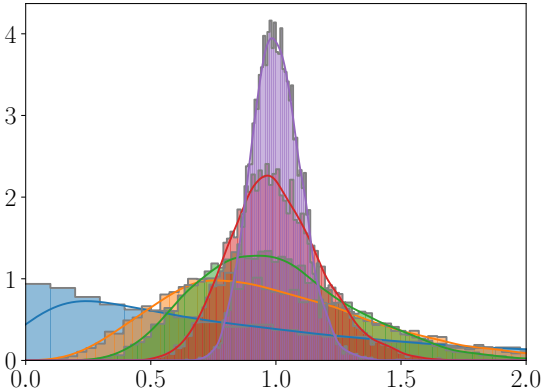

Figure: Distribution of M_n as n increases ($n = \{1, 5, 10, 30, 100\}$). Here, the sequence of random variables X_i is exponentially distributed with rate 1.

Accuracy of Monte Carlo: error estimation I

- Assuming that M_n satisfies the CLT, for some $x > 0$

$$\lim_{n\to\infty} \mathbb{P}\left[-x \leq \sqrt{n}\frac{M_n - \mu}{\sigma} \leq x\right] = \mathbb{P}[-x \leq U \leq x] \tag{15a}$$

$$\lim_{n\to\infty} \mathbb{P}\left[M_n - x\frac{\sigma}{\sqrt{\mu}} \leq \mu \leq M_n + x\frac{\sigma}{\sqrt{\mu}}\right] = \mathbb{P}[-x \leq U \leq x] = 2\Phi(-x), \tag{15b}$$

  where $U \sim \mathcal{N}(0, 1)$.

- A **confidence interval** (CI) gives information about the probability that the CI bounds contain the truth μ. Such probability is $1 - \delta$.

- Particularly, the $\delta = 95\%$ CI:

$$\mathsf{CI}_{95} = \widehat{\mu}_n \pm 1.96\frac{\sigma}{\sqrt{n}}. \tag{16}$$

Accuracy of Monte Carlo: error estimation II

- If the variance is not available. We can employ a MC estimation of the variance to get the confidence intervals. The most commonly used estimates are:

$$\widehat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \widehat{\mu}_n)^2 \quad \text{or} \quad s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \widehat{\mu}_n)^2. \tag{17}$$

- The sample variance at the left is a biased estimator of the variance. Multiplying the uncorrected sample variance by the factor $n/(n-1)$ gives the unbiased estimator at the right[3]. Generally this approach reduces the bias due to finite sample size.

- The MC confidence intervals are often better represented using s instead of $\widehat{\sigma}$.

[3] This is known as Bessel's correction.

Further considerations: displaying MC results

- Kernel density estimation (KDE) is a non-parametric method to estimate the PDF of a RV based on kernels (kernel smoothing). Given samples $\{x_i\}_{i=1}^n$ from a distribution of interest, we can estimate the shape of its PDF π via KDE as:

$$\pi(x) \approx \widehat{\pi}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} k\Big(\frac{x - x_i}{h}\Big), \tag{18}$$

  where k is the kernel and $h > 0$ is a smoothing parameter called the *bandwidth*.

- A histogram is an approximate representation of the distribution of samples: we divide the entire range of values into a series of intervals (bins) and then count how many values fall into each interval. There is no "best number of bins", and different bin sizes can reveal different features. A standard one is the *square-root choice* $\lceil \sqrt{n} \rceil$.

- My suggestion: a histogram is better than a KDE :-)

Further considerations: when MC fails

- MC methods are extremely robust (dim and type of problem).
- Problems appear when μ (or σ^2) does not exist. Hence, we typically need that $\mathbb{E}[|X|] < \infty$.
- Plots of M_n for different n, show that "*once in a while*" there are sharp jumps.
- Sometimes, re-parametrizations help to address this issue.

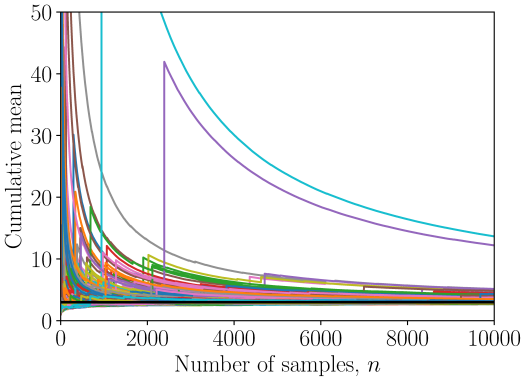

Figure: Cumulative averages for a distribution with non-defined mean.

PART IIa: simulating random numbers

Random numbers: intro I

- Monte Carlo methods require a "source of randomness".
- Random numbers are delivered as a stream of independent random variables $\mathcal{U}(0,1)$.
- Devices are built for generating true random numbers from physical processes such as radioactive particle emission, that are thought to be truly random.

Random numbers: intro I

- Monte Carlo methods require a "source of randomness".
- Random numbers are delivered as a stream of independent random variables $\mathcal{U}(0, 1)$.
- Devices are built for generating random numbers from physical processes such as radioactive particle emission, that are thought to be truly random.
- **Some drawbacks**: (i) difficult to rerun a simulation; (ii) truly random sequences cannot be compressed, a lot of storage is required; (iii) slow generation; (iv) might fail some tests for randomness (hardware acquisition).
- `RANDOM.ORG` offers true random numbers to anyone on the Internet (See link). The randomness comes from atmospheric noise.

Random numbers: intro II

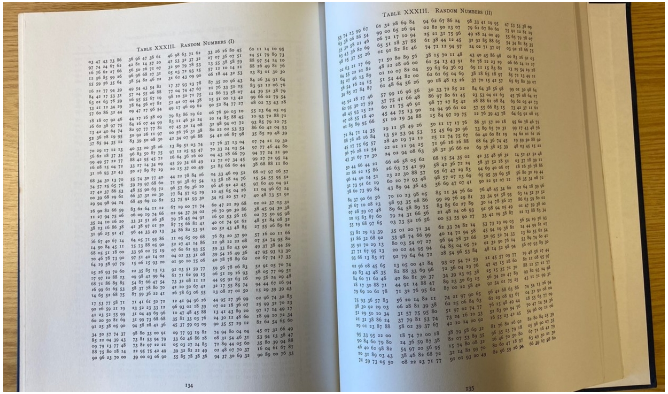

Figure: Back in the day, statistical tables provided lists of random numbers[4].

[4] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research*. 6th ed. Oliver and Boyd, 1938,

Random number generation I

- Pseudo-random numbers: based on numerical algorithms.
- These dominate the practice, we simply refer to them as random numbers.
- No random number generator perfectly simulates genuine randomness, so there is the possibility that some flaw in the generator will give a misleading Monte Carlo answer.

Random number generation I

- Pseudo-random numbers: based on numerical algorithms.
- These dominate the practice, we simply refer to them as random numbers.
- No random number generator perfectly simulates genuine randomness, so there is the possibility that some flaw in the generator will give a misleading Monte Carlo answer.
- Among these high quality generators, the Mersenne twister, MT19937, is the most popular[5].
- Sometimes a very bad random number generator will be embedded in general purpose software[6]. RANDU was a popular one in the 60's.

---

[5] M. Matsumoto and T. Nishimura. "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator". In: *ACM Trans. Model. Comput. Simul.* 8.1 (1998), 3–30.

[6] P. L'Ecuyer and R. Simard. "TestU01: A C Library for Empirical Testing of Random Number Generators". In: *ACM Trans. Math. Softw.* 33.4 (2007).

Random number generation II

- The most widely used random number generators for Monte Carlo sampling use simple recursions.
- The typical random number generator provides a function, with a name such as `rand`, that can be invoked via an assignment like `x = rand()`. What is normally inside:
  - **State**: `state = update(state)`.
  - **Period**: index P at which the generator is a deterministic cycle (i.e., $x_{i+P} = x_i$).
  - **Seed**: make a random number generator repeatable, `setseed(seed)`.

Random number generation II

- The most widely used random number generators for Monte Carlo sampling use simple recursions.
- The typical random number generator provides a function, with a name such as `rand`, that can be invoked via an assignment like `x = rand()`. What is normally inside:
  - **State**: `state = update(state)`.
  - **Period**: index P at which the generator is a deterministic cycle (i.e., $x_{i+P} = x_i$).
  - **Seed**: make a random number generator repeatable, `setseed(seed)`.
- By controlling the seed, we can synchronize multiple simulations.
- In moderately complicated simulations, we want to have two or more streams of random numbers. Constant switching between streams is cumbersome and it is prone to errors.
- While the field of random number generators is mature, design for parallel applications is still an active area.

Random number generation: linear congruential generator (LCG)

- The majority of modern random number generators are based on simple recursions using modular arithmetic.
- The class of LCG methods generate a deterministic stream of numbers using the formula:

$$x_{i+1} = (ax_i + c) \bmod m, \tag{19}$$

  where $0 < m$ is the modulus, $0 < a < m$ is the multiplier, $0 \leq c < m$ is the increment, and $0 \leq x_0 < m$ is the seed.

- If $c = 0$, the generator is often called a multiplicative congruential generator (MCG), or Lehmer RNG[7].
- Some standard parameter values: $m = 232, a = 1664525, c = 1013904223$ [7].

[7] D. H. Lehmer. "Mathematical methods in large-scale computing units". In: *Proceedings of 2nd Symposium on Large-Scale Digital Calculating Machinery*. 1951, 141–146.

Random number generation: other linear generators

- *Multiple recursive generators* (MRG) of order k, determined by a sequence of k-dimensional state vectors $\boldsymbol{x} = [x_{t-k+1}, \ldots, x_t]$, for $t = 0, 1, \ldots$, whose components satisfy the linear recurrence:

$$x_{i+1} = (a_1 x_{t-1} + \cdots + a_k x_{t-k}) \bmod m, \qquad t = k, k+1, \ldots \tag{20}$$

- *Lagged Fibonacci generators*, which take the form $xi = (x_{i-r} + x_{i-s}) \bmod m$ for carefully chosen r, s and m are an important case because they are fast.

- *Inversive congruential generator* (ICG), for a prime number m, the ICG update is:

$$x_{i+1} = \begin{cases} (a_0 + a_1 x_{i-1}^{-1}) \bmod m & x_{i-1} \neq 0 \\ a_0 & x_{i-1} = 0. \end{cases} \tag{21}$$

One generator to rule them all: Mersenne Twister

- There are three main ways to choose m. Sometimes it is advantageous to choose m to be a large prime number[8]. Another choice is to take $m = 2^r$ for some integer $r > 1$. The third choice, with $m = 2$ is convenient because it allows fast operations.

- The most widely used modulo 2 generator is the Mersenne Twister (MT) [5]. The MT algorithm is based on a matrix linear recurrence over a finite binary field. It commonly uses $m = 219937 - 1$ (MT19937).

[8] A prime number of the form $2k - 1$ is called a Mersenne prime (after Marin Mersenne (1588-1648), a french polymath).

One generator to rule them all: Mersenne Twister

- There are three main ways to choose m. Sometimes it is advantageous to choose m to be a large prime number. Another choice is to take $m = 2^r$ for some integer $r > 1$. The third choice, with $m = 2$ is convenient because it allows fast operations.

- The most widely used modulo 2 generator is the Mersenne Twister (MT) [4]. The MT algorithm is based on a matrix linear recurrence over a finite binary field. It commonly uses $m = 219937 - 1$ (MT19937).

- It passes numerous tests for statistical randomness, including the Diehard tests[9] and most of the TestU01 tests.

- **Alternatives**: WELL (Well Equidistributed Long-period Linear), ACORN (Additive Congruential Random Number), PCG (permuted congruential generator).

[9] Yes, like in the movie. There is even a set of *Dieharder* tests.

Random number generation: uniformity tests

- For P large enough, the genuinely random values would have a very flat histogram with high probability. Designers of random number generators carefully choose parameters for us.
- A way that bad generators are eliminated is by insisting that the consecutive pairs (x_1, x_2), $(x_2, x_3), \ldots, (x_P, x_1)$ have approximately the $\mathcal{U}[0,1]^2$ distribution.
- Ideally, the k-tuples $(x_i, x_{i+1}, \ldots, x_{i+k-1})$ for general k should also be uniform, at least for the smaller values of k. **Note**: *the random number generators we use in practice can only guarantee uniformity up to some moderately large values of* k. For example, the MT19937 is $k = 623$-distributed to 32 bits accuracy.
- It has been shown that the consecutive tuples $(x_i, x_{i+1}, \ldots, x_{i+k-1})$ from an MCG have a lattice structure.
- Some statistical tests have been designed for small subsets of random number generator output, e.g., p values in a χ^2 or Kolmogorov–Smirnov tests.

Random number generation: uniformity tests

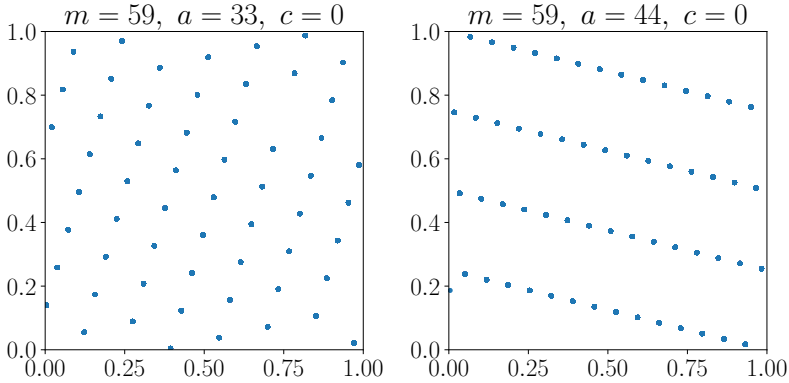

Figure: MCG: tuples for $k = 2$ arrange in a lattice structure. Which one is better? The distance between the parallel lines is a measure of uniformity; one seeks to minimize the maximum separation.

Random number generation: example 1

- Let us compute the following integral by Monte Carlo:

$$I = \int_0^2 \int_1^2 \int_0^\pi xy \sin(z)\, \mathrm{d}x\mathrm{d}y\mathrm{d}z. \tag{22}$$

- The analytical solution is 6. The MC estimate is:

$$I \approx \widehat{I} = \frac{V}{n} \sum_{i=1}^{n} f(x_i, y_i, z_i), \tag{23}$$

with standard deviation

$$V \frac{\sigma}{\sqrt{n}}. \tag{24}$$

Random number generation: example 2

Let us estimate π by Monte Carlo:

- To estimate the method consists of drawing a square with an inner circle. We know the area of the circle is $A_\circ = \pi r^2$ and the are of the square is $A_\square = 4r^2$.
- Then we divide the area of the circle, by the area of the square to get $A_\circ / A_\square = \pi/4$. Hence, $\pi = 4A_\circ / A_\square$.
- If a large number of random points inside the square is generated and the quantity of points inside the circle is counted. We can use the following ratio to estimate:

$$\pi \approx 4 \frac{\text{number of samples in the circle}}{\text{number of samples in the square}}. \tag{25}$$

Random number generation: example 2

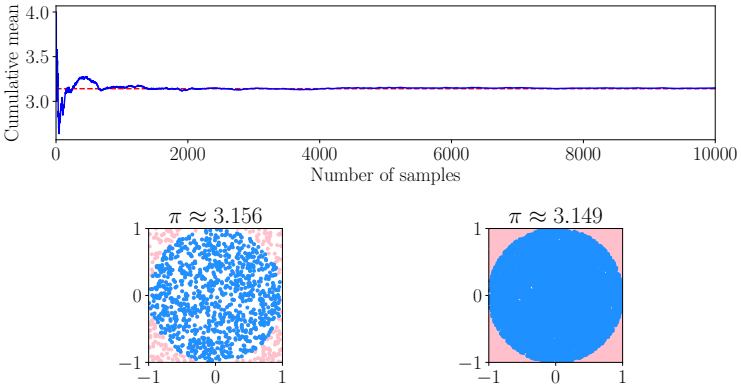

Figure: MC estimation of π. Top: evolution of the estimate for different samples. Bottom left: estimate at $n = 10^3$. Bottom right: estimate at $n = 10^4$

PART IIb: simulating random variables

Random variable generation

- Uniform random variables are the basic building block for Monte Carlo methods.
- Most widely used mathematical computing environments include generators for a wide selection of non-uniform distributions. For `numpy.random` (see Link).
- However, sometimes we need to sample from a distribution that our environment does not include. Hence, we need to understand how non-uniform random numbers are generated.
- We discuss the most popular methods in the next.

Random variable generation: inverse transform method I

- The most direct way to convert uniform into non-uniform random variables is by inverting the CDF.
- Suppose that the random variable X has PDF $\pi_X(x)$ for all $x \in \mathbb{R}$, then $F_X(x)$ is monotone and continuous and it has an inverse

$$F_X^{-1}(u) = \inf(x \in \mathbb{R} : F_X(x) \geq u), \quad (26)$$

for a probability $0 < u < 1$, called the quantile function.

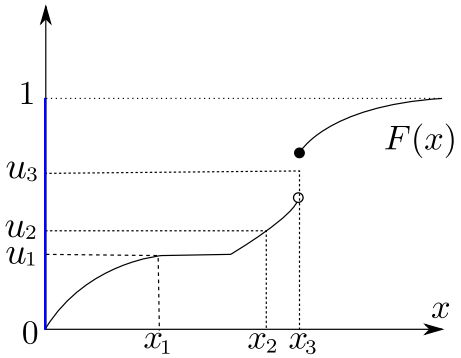

Figure: Recall that the quantile function is the CDF inverse.

Random variable generation: inverse transform method II

- We want to find some strictly monotone transformation $T : [0, 1] \to \mathbb{R}$, such that $T(U) \stackrel{d}{=} F_X$.
- If we have $U \sim \mathcal{U}(0, 1)$ and define $X = T(U)$, then

$$F_X(x) = \mathbb{P}[X \leq x] = \mathbb{P}[T(U) \leq x] = \mathbb{P}\left[U \leq T^{-1}(X)\right] = T^{-1}(X) \quad x \in \mathbb{R}, \tag{27}$$

  in the last step used that $\mathbb{P}[U \leq u] = u$ since U is standard uniform. Therefore, we have $T(u) = F_X^{-1}(u), u \in [0, 1]$.

Random variable generation: inverse transform method II

- We want to find some strictly monotone transformation $T : [0, 1] \to \mathbb{R}$, such that $T(U) \overset{d}{=} F_X$.
- If we have $U \sim \mathcal{U}(0, 1)$ and define $X = T(U)$, then

$$F_X(x) = \mathbb{P}[X \leq x] = \mathbb{P}[T(U) \leq x] = \mathbb{P}\left[U \leq T^{-1}(X)\right] = T^{-1}(X) \quad x \in \mathbb{R}, \tag{25}$$

  in the last step used that $\mathbb{P}[U \leq u] = u$ since U is standard uniform. Therefore, we have $T(u) = F_X^{-1}(u), u \in [0, 1]$.

- The inverse transform method simply consist of[10]:
  - Drawing a random number $u \sim \mathcal{U}(0, 1)$.
  - Computing $x = F_X^{-1}(u)$.

[10] I encourage you to check the examples in section 4.2 of Owen's book.

Random variable generation: inverse transform method III

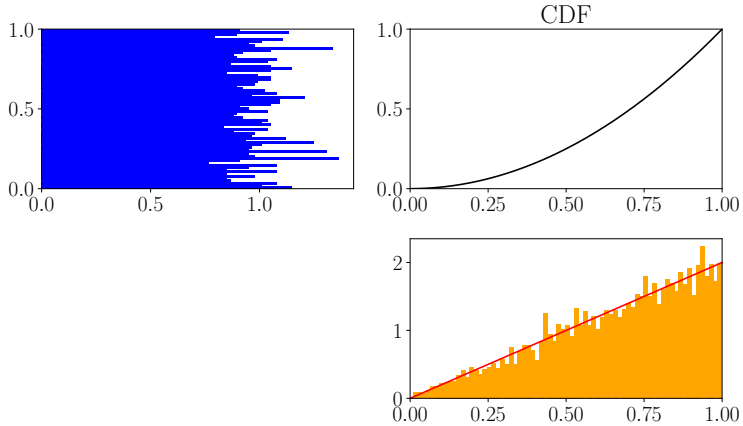

Figure: Example: inverse transform method for the density $\pi_X(x) = 2x$.

Random variable generation: inverse transform method IV

- For the Gaussian distribution, the CDF and quantile function do not exist in closed form. There are multiple methods that estimate them in an efficient way (see, e.g., section 4.3 in Owen's).

- In some problems, we have no direct F_X^{-1} available but we might be able to invert F_X numerically, if we know the PDF (or the characteristic function).

- Sometimes we get a better result from more general (monotone) transformations. These typically depend on specialized relationships among distributions. As we show in the next.

Random variable generation: acceptance-rejection method I

- Sometimes we cannot find a computable transformation that takes a random number and produces a random variable X with the distribution F_X that we want.
- We can sample from another distribution with density q, and by strategically rejecting some values from q while accepting the others, we produce the desired distributed values.
- Rejection sampling $=$ accept-reject sampling $=$ acceptance-rejection sampling.
- We present the idea, assuming continuous distributions with PDFs π_X and q, respectively. The method extends to discrete distributions as well.

Random variable generation: acceptance-rejection method II

- To use acceptance-rejection, we find a proposal density $q(x)$ and a finite covering constant c, satisfying three conditions:
  - we can "easily" sample q;
  - the ratio $\pi_X(x)/q(x)$ is well-defined (bounded);
  - and $\pi(x) \leq c\, q(x)$ with $\text{supp}(\pi_X(x)) \subseteq \text{supp}(q(x))$.

- Hence, we repeatedly sample candidates $x^\star \sim q$. The candidate is accepted with probability

$$\alpha = \frac{\pi_X(x^\star)}{c\, q(x^\star)} \tag{28}$$

otherwise it is rejected and we try a next candidate.

Random variable generation: acceptance-rejection method III

`setseed`(seed)

Algorithm 1: Version 1: standard.

Generate $x^\star \sim q$;
Compute the acceptance probability $\alpha = \pi_X(x^\star)/c\,q(x^\star)$;
Generate $u \sim \mathcal{U}(0,1)$;
if $u \leq \alpha$ **then**
    $x \leftarrow x^\star$;
end

Algorithm 2: Version 2: geometric.

Generate $x^\star \sim q$;
Generate $y \sim \mathcal{U}(0, c\,q(x^\star))$;
if $y \leq \pi_X(x^\star)$ **then**
    $x \leftarrow x^\star$;
end

Random variable generation: acceptance-rejection method IV

- Selection of c: the covering constant can be selected such that $c \geq \sup_x(\pi_X(x)/q(x))$. In practice, we would want c as close to 1 as possible (i.e., $\pi_X(x) \approx q(x)$).
- The number of proposals we must make before one is accepted has a geometric distribution with $\mathbb{P}[N = k] = c^{-1}(1 - c^{-1})^{k-1}$. As a result $\mathbb{E}[N] = c$. **Hence**: *the expected number of iterations required until a sample is successfully generated is* c.

Random variable generation: acceptance-rejection method IV

- Selection of c: the covering constant can be selected such that $c \geq \sup_x(\pi_X(x)/q(x))$. In practice, we would want c as close to 1 as possible (i.e., $\pi_X(x) \approx q(x)$).
- The number of proposals we must make before one is accepted has a geometric distribution with $\mathbb{P}[N = k] = c^{-1}(1 - c^{-1})^{k-1}$. As a result $\mathbb{E}[N] = c$. **Hence**: *the expected number of iterations required until a sample is successfully generated is* c.
- Geometric interpretation: if we sample a point (X, Y) uniformly in the portion of the plane between the horizontal axis and the curve $\pi_X(x)$, then the marginal distribution of X is π_X. To get such points, we can sample uniformly from the region under the curve $cq(x)$ and keep only the points below π_X. The sets have the form:

$$\mathcal{S}_c(q) = \{(x, y) : 0 \leq y \leq cq(x),\, x \in \mathbb{R}\}. \tag{29}$$

Random variable generation: acceptance-rejection method V

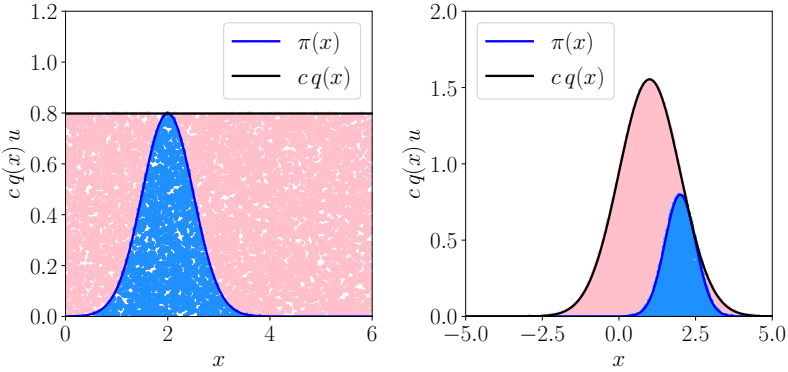

Figure: Example: acceptance-rejection method for a Gaussian density. Left: using an uniform proposal ($c \approx 4.79$, acc $= 0.21$). Right: using a Gaussian proposal ($c \approx 3.89$, acc $= 0.26$).

Random variable generation: rectangle-wedge-tail (rwt) method

- We approximate π_X by a mixture distribution whose components are rectangles, wedges, and one or two tails:

$$X \sim \pi_X(\cdot;\theta), \qquad \theta \sim \text{rwt}.$$

- To sample from π_X, we choose a component at random and sample from it:

  If rectangle: $\mathcal{U}(a,b)$; if wedge on $[a,b]$:

$$\frac{\phi(x)-\phi(b)}{\Phi(b)-\Phi(a)-(b-a)\phi(b)};$$

  if tail on $[B,\infty)$: $\phi(x)/(1-\Phi(B))$.

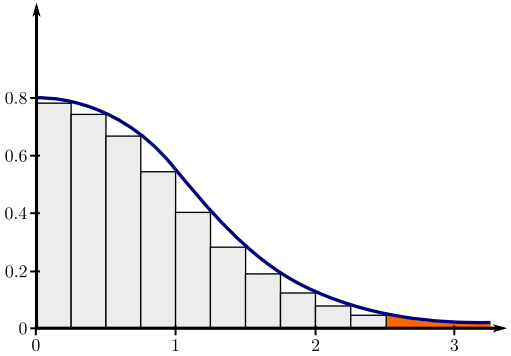

Figure: rwt scheme: 10 rectangles, 10 wedges and 1 tail.

One generator to rule them all: Ziggurat method

- For the half-normal π_X, it generates a sample point (X, Y) uniformly distributed in

$$\mathcal{S}_c(\pi_X) = \{(x, y) : 0 \leq y \leq \exp(-x^2/2)\}$$

with $c = \sqrt{\pi/2}$.

- When a horizontal rectangle, we do acceptance-rejection. When the bottom region is chosen, it can be sampled as a mixture of a horizontal subregion and a tail.

- The ziggurat method is very fast. Setting up for it requires solving a set of equations [4].

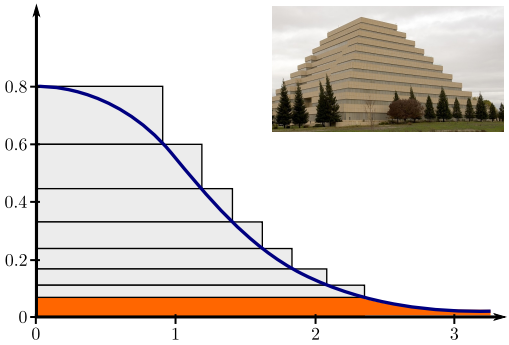

Figure: Ziggurat scheme: the method typically has 128 or 256 regions.

Random vector generation: multivariate Gaussian

- We will often encounter the task of simulating $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, or using the precision matrix $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$.
- Note that the only distribution we actually sample from is a 1D Gaussian distribution!

Algorithm 3: Version 2: with mean and covariance matrix.

1 Compute Cholesky factorization, $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$;
2 Generate $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$;
3 Compute $\boldsymbol{x} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{z}$;

Algorithm 4: Version 2: with mean and precision matrix.

1 Compute Cholesky factorization, $\boldsymbol{\Lambda} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$;
2 Generate $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$;
3 Solve $\mathbf{L}^{\mathsf{T}}\boldsymbol{y} = \boldsymbol{z}$;
4 Compute $\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{y}$;

Random variable generation: example 1

Let us generate a sample from the exponential distribution.

- If $X \sim \text{Exp}(\lambda)$, the CDF has the form:

$$F_X(x) = 1 - \exp(-\lambda x), \qquad x \geq 0, \lambda > 0. \tag{30}$$

- Then, solving $u = F_X(x)$:

$$x = F_X^{-1}(u) = -\frac{1}{\lambda}\log(1-u). \tag{31}$$

- Note that $u \sim \mathcal{U}(0,1)$ implies that $(1-u) \sim \mathcal{U}(0,1)$.

Random variable generation: example 2

Let us estimate the Euler–Mascheroni constant γ by Monte Carlo:

- Set $X \sim \text{Exp}(1)$. We can write the constant as:

$$\gamma = -\int_0^\infty \exp(-x)\log(x)\,\mathrm{d}x = \int_0^\infty -\log(x)\pi_X(x)\,\mathrm{d}x = \mathbb{E}[-\log(x)]\,. \tag{32}$$

- Then, we can draw $\{x_i\}_{i=1}^n \overset{\text{idd}}{\sim} \text{Exp}(1)$ and

$$\gamma \approx \widehat{\gamma}_n = -\frac{1}{n}\sum_{i=1}^n \log(x_i). \tag{33}$$

Random variable generation: final comments

- Random number and variable generation is a large field of numerical analysis. There exist many other methodologies we did not cover here.
- Most of the methods presented in Part II are incorporated into statistical toolboxes, e.g., `numpy.random` or the `rvs` method in `scipy.stats`.
- Random number and variable generation is the core of UQ. We will see that doing UQ for inverse problems basically amounts to sampling.

Random variable generation: final comments

- Random number and variable generation is a large field of numerical analysis. There exist many other methodologies we did not cover here.
- Most of the methods presented in Part II are incorporated into statistical toolboxes, e.g., `numpy.random` or the `rvs` method in `scipy.stats`.
- Random number and variable generation is the core of UQ. We will see that doing UQ for inverse problems basically amounts to sampling.
- The acceptance-rejection method also works for dimension d larger than 1. The issue, however, is that it grows inefficient with d. The method provides the foundation to Markov chain MC methods, such as Metropolis–Hastings.
- The standard MC method requires samples that are independent and identically distributed. We will see what happens when samples are correlated later in the course.

References

[1] R. A. Fisher et al. *Statistical tables for biological, agricultural and medical research*. 6th ed. Oliver and Boyd, 1938, p. 98.

[2] P. L'Ecuyer et al. "TestU01: A C Library for Empirical Testing of Random Number Generators". In: *ACM Trans. Math. Softw.* 33.4 (2007).

[3] D. H. Lehmer. "Mathematical methods in large-scale computing units". In: *Proceedings of 2nd Symposium on Large-Scale Digital Calculating Machinery*. 1951, 141–146.

[4] G. Marsaglia et al. "The Ziggurat method for generating random variables". In: *Journal of Statistical Software* 5.8 (2000).

[5] M. Matsumoto et al. "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator". In: *ACM Trans. Model. Comput. Simul.* 8.1 (1998), 3–30.

[6] A. B. Owen. *Monte Carlo theory, methods and examples*. `artowen.su.domains/mc/`, 2018.

[7] W. H. Press et al. *Numerical recipes in C: the art of scientific computing*. 3rd ed. Cambridge University Press, 2007.

[8] J. S. Rosenthal. *A first look at rigorous probability theory*. 2nd ed. World Scientific Publishing Company, 2006.