Monte Carlo method

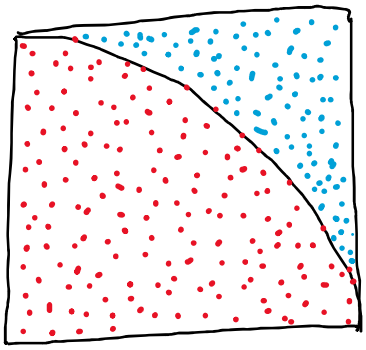
domenica 14 settembre 2025 10:53

Monte Carlo methods represent a versatile group of computational techniques that rely on repeated random sampling to produce numerical results. The core idea is to harness randomness to tackle problems that, while deterministic in theory, may be too complex to solve directly. Typically implemented through computer simulations, these methods offer approximate solutions to challenges that are otherwise analytically intractable.

What is it?

Their main applications fall into three categories: optimization, numerical integration, and sampling from probability distributions. Additionally, Monte Carlo methods are valuable for modeling systems with high levels of uncertainty, such as assessing the probability of rare but critical events—for instance, estimating the risk of a nuclear power plant failure.

example



As an illustration, consider a quadrant (a quarter of a circle) inscribed within a unit square. Since the ratio of their areas is $\pi/4$, this relationship can be used to approximate the value of π through the Monte Carlo method:

1. Draw a unit square and inscribe a quarter circle (radius = 1) inside it.
2. Randomly and uniformly scatter a set of points across the square.
3. For each point, determine whether it falls inside the quarter circle. This is done by calculating its distance from the origin using the Pythagorean theorem:

$$d = \sqrt{x^2 + y^2}$$

   If $d \leq 1$, the point lies inside the quarter circle; if $d > 1$, it lies outside but still within the square.
4. The ratio between the number of points inside the quarter circle and the total number of points approximates the area ratio $\pi/4$.
5. Multiplying this ratio by 4 yields an estimate of π.

Monte Carlo methods work because they are grounded in the **Law of Large Numbers**. This fundamental theorem in probability theory states that as the number of independent trials of a random experiment increases, the average of the observed outcomes converges to the expected value.

yes, but why do they work?

In simpler terms, the more random samples we generate, the closer our empirical estimate will be to the true probability or numerical value we are trying to approximate.

Theorem (Law of Large Numbers):
Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with finite expected value $\mu = \mathbb{E}[X_i]$. Then:

$$\frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{n\to\infty} \mu \quad \text{(with probability 1)}.$$

This result justifies why increasing the number of sampled points in a Monte Carlo simulation improves the accuracy of the approximation.

this is an approximation so... what is the error?

While the Law of Large Numbers explains why Monte Carlo methods converge to the correct result, it does not describe *how fast* this convergence happens. For this purpose, we rely on the **Central Limit Theorem (CLT)**, which characterizes the error of the approximation.

The CLT states that, given a large number of independent and identically distributed random variables, their normalized average will follow a normal distribution centered around the true expected value.

Theorem (Central Limit Theorem):
Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed random variables with expected value $\mu = \mathbb{E}[X_i]$ and finite variance $\sigma^2 = \mathrm{Var}(X_i)$. Then:

$$\sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right) \xrightarrow{d} \mathcal{N}(0, \sigma^2),$$

where $\xrightarrow{d}$ denotes convergence in distribution.

This means that the error in a Monte Carlo approximation decreases proportionally to $\frac{1}{\sqrt{n}}$. In practice, to reduce the error by a factor of 10, the number of samples must be increased by a factor of 100.

In other words $\lim_{n \to +\infty} \mathbb{P}\left(\frac{X_1 + X_2 + \dots + X_n - n\mu}{\sqrt{n}\,\sigma} \leq x \right) = \phi(x)$

where $\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-s^2/2}\, ds$

↝ as a consequence of this theorem, we obtain for a large n:

$$\mathbb{P}\left(\tilde{X}_n - z_{\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{n}} < \mu < \tilde{X}_n + z_{\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{n}} \right) = 1 - \alpha$$

$\Downarrow$

$\left(\tilde{X}_n - z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}} ; \tilde{X}_n + z_{\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{n}} \right)$ is

a confidence interval of level α

$\Downarrow$

what if we don't know σ? Then, we can use the empirical variance

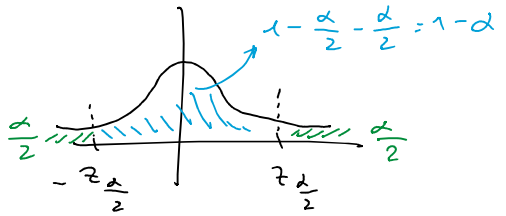
$$S^2 = \sum_{i=1}^{n} \frac{(X_i - \tilde{X}_n)^2}{n-1}$$

which is a correct estimator for the variance

(a.k.a. $\mathbb{E}(S^2) = \sigma^2$)

NOTES:

1)



2) $\tilde{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$

for coding

In practice, Python provides several libraries that make Monte Carlo simulations more powerful and efficient:

- **NumPy**: for fast generation of random numbers, vectorized operations, and efficient array handling.
- **SciPy**: offers probability distributions, statistical tools, and integration routines that can support more advanced Monte Carlo methods.
- **random** (standard library): simple random number generation for small-scale or educational simulations.
- **PyMC** and **PyMC3/4**: probabilistic programming libraries that implement Bayesian inference using advanced Monte Carlo techniques such as Markov Chain Monte Carlo (MCMC).
- **TensorFlow Probability** and **PyTorch distributions**: provide Monte Carlo and MCMC methods integrated with deep learning frameworks.
- **SALib**: useful for sensitivity analysis and uncertainty quantification, often combined with Monte Carlo sampling.

remember ?

```python
import numpy as np

# Number of random points
n_samples = 1_000_000

# Generate random (x, y) points in the unit square
x = np.random.rand(n_samples)
y = np.random.rand(n_samples)

# Check how many points fall inside the quarter circle
inside = (x**2 + y**2) <= 1

# Monte Carlo estimate of π
pi_estimate = 4 * np.sum(inside) / n_samples

print("Estimated π:", pi_estimate)
```

other applications

Physics

- **Statistical mechanics**: simulating particle systems, Ising models, and phase transitions.
- **Quantum physics**: path integral methods, quantum Monte Carlo for solving many-body problems.
- **Radiation transport**: modeling how photons, neutrons, or other particles interact with matter.
- **Astrophysics**: estimating stellar evolution processes and cosmic ray propagation.

Engineering

- **Reliability analysis**: predicting failure rates in complex systems (e.g., power plants, bridges).
- **Risk assessment**: safety evaluations under uncertain input conditions.
- **Optimization**: solving high-dimensional design optimization problems.
- **Signal processing**: Monte Carlo simulations for filtering and noise reduction.

Finance & Economics

- **Option pricing**: estimating the value of financial derivatives (e.g., Black–Scholes models).
- **Portfolio optimization**: assessing risk-return trade-offs under uncertainty.
- **Value at Risk (VaR)**: calculating risk exposure for banks and investment funds.
- **Macroeconomic modeling**: forecasting under stochastic shocks.

Computer Science & Machine Learning

- **Probabilistic inference**: Markov Chain Monte Carlo (MCMC) for Bayesian models.
- **Reinforcement learning**: Monte Carlo control methods for policy evaluation.
- **Graphics & rendering**: simulating realistic lighting and global illumination.
- **Algorithm testing**: randomized algorithms and simulation-based performance analysis.

Biology & Medicine

- **Population genetics**: simulating evolutionary dynamics.
- **Epidemiology**: modeling disease spread under uncertain parameters.
- **Medical imaging**: photon transport simulations in tissue for CT, PET, or MRI scans.
- **Drug discovery**: exploring molecular conformations and binding affinities.

Other Fields

- **Environmental science**: climate modeling and uncertainty quantification.
- **Operations research**: scheduling, queuing theory, and logistics optimization.
- **Game theory & decision-making**: simulating strategies under uncertainty.