Probabilistic optimization in manufacturing

Simulated Annealing meets Set Packing

Michal Racko
PyCon PL 2025

August 30, 2025

https://github.com/michal-racko/pycon_pl_2025

What are Monte Carlo methods?

- Statistical techniques using random sampling
- Solve problems that are impossible or impractical to solve analytically
- Key principle: Use randomness to approximate deterministic results

Applications:

- Physics simulations
- Financial modeling
- Machine learning
- Engineering optimization

Starting at square one

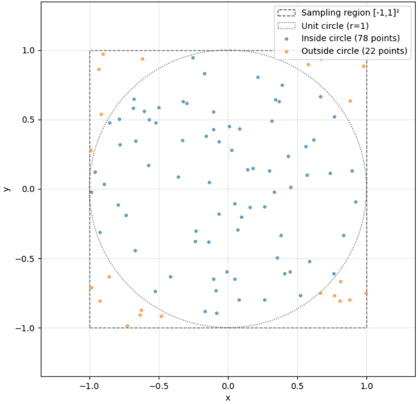
Value of π can be estimated using random sampling

Let's pretend π is an unknown constant which has to be estimated.

Geometric considerations

- Circle area: $A_{circle} = \pi$
- Square area: $A_{square} = 4$
- Ratio: $\frac{\pi}{4} = \frac{A_{circle}}{A_{square}}$

Therefore: $\pi = 4 \times \frac{A_{circle}}{A_{square}}$



Unit circle inscribed in square

Starting at square one

Value of π can be estimated using random sampling

Key Insight: All random points are uniformly distributed in the square

- Point (x, y) is inside circle if: $x^2 + y^2 \leq 1$
- Point (x, y) is outside circle if: $x^2 + y^2 > 1$

Therefore we can estimate

$$\pi \approx 4 \times \frac{\text{points inside circle}}{\text{total points}}$$

Starting at square one

Value of π can be estimated using random sampling

```
>>> import numpy as np
>>> class MonteCarloSamples:
...     def __init__(self, n_samples: int):
...         # Generate random points in [-1,1] x [-1,1]
...         self._samples = np.random.random((n_samples, 2)) * 2 - 1
...
...     def __len__(self) -> int:
...         return len(self._samples)
...
...     @property
...     def centre_distances(self) -> np.ndarray:
...         return np.sqrt((self._samples ** 2).sum(axis=1))
...
...     @property
...     def within_unit_circle(self) -> np.ndarray:
...         return self.centre_distances <= 1
...
...     @property
...     def pi_estimate(self) -> float:
...         return float(self.within_unit_circle.sum() / len(self) * 4)
...
>>> samples = MonteCarloSamples(100)
>>> print(samples.pi_estimate)
3.12
```

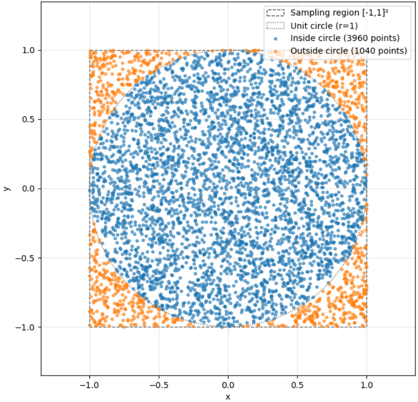
Starting at square one

Value of π can be estimated using random sampling

Adding more random samples improves precision

Geometric considerations

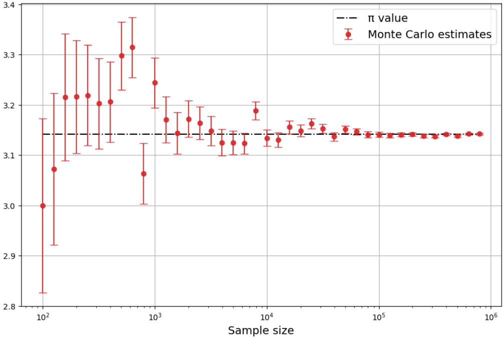
- 100 samples: $\pi \approx 3.12$
- 5,000 samples: $\pi \approx 3.1680$
- 10,000,000 samples: $\pi \approx 3.1408$



More random points drawn from the uniform distribution

Starting at square one

Value of π can be estimated using random sampling



Monte Carlo estimates converge to the true value as $N \to \infty$

Uncertainty estimation

- Our estimate follows: $X \sim \mathrm{Binomial}(N, p)$
- $\mathrm{Var}(\hat{\pi}) = 16 \cdot \frac{p(1-p)}{N}$
- Standard deviation: $\sigma \propto \frac{1}{\sqrt{N}}$

S&P 500 price prediction

Monte Carlo can model complex or poorly understood processes

Estimating via Random Sampling: Random sampling can estimate quantities of interest when the experimental setup is well designed

Sample Size vs. Error: Estimation error decreases as sample size grows, although larger sample sizes increase computational complexity

Quantifying Uncertainty: Uncertainty can be inferred using the properties of the probability distribution underlying the random samples

S&P 500 price prediction

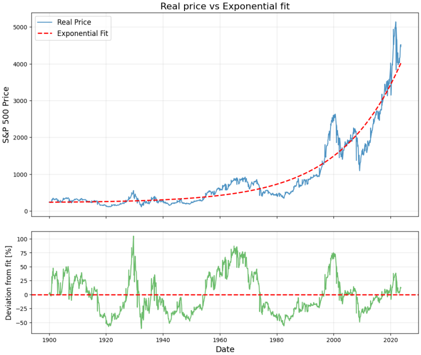
Monte Carlo can model complex or poorly understood processes

Let's divide and conquer

Decompose timeseries

- Exponential Trend: $P_{\mathrm{e}}(t) = a \cdot e^{bt} + c$
- Brownian motion: $\Delta P_{\mathrm{b}}(t) = \frac{P(t) - P_{\mathrm{e}}(t)}{P_{\mathrm{e}}(t)}$

$$P(t) = P_{\mathrm{e}}(t) \times (1 + \Delta P_{\mathrm{b}}(t))$$



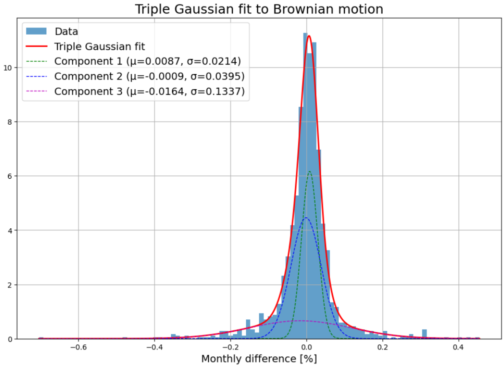
Exponential growth of S&P 500

S&P 500 price prediction

Monte Carlo can model complex or poorly understood processes

Triple Gaussian mixture:

$$f(x) = a_1 \cdot \mathcal{N}(\mu_1, \sigma_1^2) + a_2 \cdot \mathcal{N}(\mu_2, \sigma_2^2) + a_3 \cdot \mathcal{N}(\mu_3, \sigma_3^2)$$



Different regimes

- Normal market conditions
- Market stress/crashes
- Market euphoria/bubbles

S&P 500 price prediction

Monte Carlo can model complex or poorly understood processes

Now draw random samples from the fitted distribution

```
>>> import numpy as np
>>> N_SAMPLES = 10_000
>>> total_weight = a1 + a2 + a3
>>> component_choice = np.random.random(N_SAMPLES)
>>> samples = np.where(
...     component_choice < a1 / total_weight,
...     np.random.normal(mu1, sigma1, N_SAMPLES),
...     np.where(
...         component_choice < (a1 + a2) / total_weight,
...         np.random.normal(mu2, sigma2, N_SAMPLES),
...         np.random.normal(mu3, sigma3, N_SAMPLES)
...     )
... )
>>> samples
array([-0.00252675,  0.15553425,  0.0344586 , ...,  0.01754364,
       -0.01048925, -0.01011193], shape=(10000,))
```

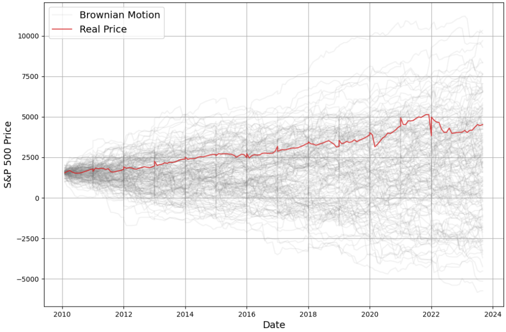
and repeat this for every timestep of our simulation. . .

S&P 500 price prediction

Monte Carlo can model complex or poorly understood processes

Many experiments

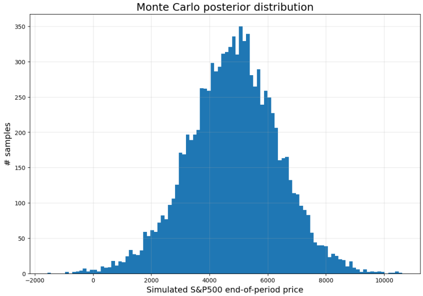
- Every simulated path represents an alternative reality following the same principles as our real data
- The more paths in a given region the more likely it is to occur
- Variance reduction techniques help focus on important regions



Simulated S&P 500 paths

S&P 500 price prediction

Monte Carlo can model complex or poorly understood processes



Monte Carlo posterior is comprised of results of all of the parallel experiments run by the model

Our simulation

- Gives probability estimates for future outcomes
- Past two years get Z-score of 1.0444
- While this model is rather simplistic it still provides useful insights

S&P 500 price prediction

Monte Carlo can model complex or poorly understood processes

Modeling with Monte Carlo: Complex or poorly understood processes can be modeled using random distributions, which Monte Carlo methods then leverage to simulate possible future outcomes

Assessing Future Outcomes: Posterior distributions allow us to estimate the likelihood of future outcomes, providing a powerful tool for assessing risks and opportunities

Statistical Weight Implementation: Variance reduction techniques allow our model to concentrate computational effort on the most important regions of the outcome space

Probabilistic Optimization

Simulated Annealing escapes local optima

The Challenge: Roadtrip around Poland

Problem complexity

- NP-hard optimization problem
- $(n-1)!/2$ possible routes for n cities
- 21 cities $\rightarrow \sim 10^{18}$ combinations

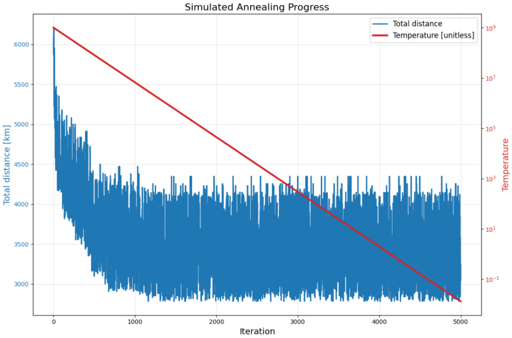
Naive solution to TSP gets trapped in a local minimum

Probabilistic Optimization

Simulated Annealing escapes local optima

Common for:

- Sheet metal production
- Quartz minerals
- Nuclear energy



Boltzmann Distribution:

$$P(E) \propto e^{-\frac{E}{k_B T}}$$

Simulated analogy:

$$P(\text{accept worse solution}) = e^{-\frac{\text{distance increase}}{T}}$$

Probabilistic Optimization

Simulated Annealing escapes local optima

Algorithm

- **Exploration phase**
  high T $\Rightarrow$ most moves are accepted
- **Exploitation phase**
  low T $\Rightarrow$ only improvements are accepted

My implementation

Batch annealing plant optimization

Simulated annealing finds approximation to the optimal solution

Probabilistic Optimization

Simulated Annealing escapes local optima

Monte Carlo in Optimization: Randomness enables exploration of solution spaces that deterministic algorithms cannot effectively navigate

Temperature Scheduling: Controlled cooling balances exploration (high temperature) with exploitation (low temperature) to find global optima

Real-world Impact: TSP principles apply to logistics, manufacturing, and many more...

Thank You!

I'm looking forward to answering your questions