Modeling and Simulation

Dr. Belkacem KHALDI
b.khaldi@esi-sba.dz
ESI-SBA, Algeria

Level: 2nd SC Class
Date:May 11, 2022

Lecture 5:

Solving Problems with Monte-Carlo Simulation Technique

What is Monte-Carlo Method

Monte-Carlo Method:

computational method using repeated random sampling to obtain numerical results.

- named after gambling in Monte-Carlo casino in Monaco.

- Technique invented in the context of the development of the atomic bomb in the 1940's.
- Provides generally approximate solutions to problems that are hard to solve analytically or numerically.
- Implemented with the uses of pseudo-random number generators

Figure 1: Monte-Carlo Casino in Monaco

Solving Problems with Monte-Carlo Simulation Technique

Monte-Carlo Domains of Applications

- Actually, widely used in:
  - Engineering,
  - Physical Sciences,
  - Computational Biology,
  - Computer Graphics,
  - Artificial intelligence for Games,
  - Finance and Business,
  - Applied statistics,
  - Project Planning,
  - ...etc.

Solving Problems with Monte-Carlo Simulation Technique

Basics of Monte-Carlo Simulation

- **Monte-Carlo (MC) Simulation:** A method of estimating the value of an unknown quantity or solving deterministic systems using the principles of inferential statistics.
- Multiple samples are collected and used to approximate the desired quantity.
  - Exactly what we did in simulating probabilities of throwing a dice (See Lecture 03: Probabilities and Random Number Simulation).

Inferential Statistics:

- Observation: Result from one trial of an experiment.
- Population: Space of all possible observations that could be seen from a trial.
- Sample: Group of results gathered from separate independent trials.
- Key fact: a random sample tends to exhibit the same properties as the population from which it is drawn.

Solving Problems with Monte-Carlo Simulation Technique

The Probabilistic Basis for Monte-Carlo: Law of Large Numbers (LLN)

Estimation of the Mean:
Given a probability space (Ω, P) and a random variable X on it, the quantity we want to estimate is the mean of X.

LLN Theorem

Let X_n be an independent, and identically distributed (i.i.d.) sequence sampled from any probability distribution $\mathcal{P}$ with mean $\mu = E[X]$. Then:

$$\hat{\mu}_n = \lim_{n \to \infty} \left(\frac{1}{n} \sum_{k=1}^{n} X_k \right) \to \mu$$

Important Note: the more random trials that are performed, the more accurate the approximated quantity, $\hat{\mu}_n$, will become.

The Probabilistic Basis for Monte-Carlo: Law of Large Numbers (LLN)

Error Estimation in $\hat{\mu}_n$:

- The LLN tells us that $\hat{\mu}_n$ converges to μ but it does not tell us anything about how close $\hat{\mu}_n$ is to μ.
- In any Monte Carlo simulation we do not actually let $n \to \infty$, we only use a (hopefully) large value of n.
- So it is crucial to address this question of how close our approximation is.

In Statistics, theory tells us that the difference of $\hat{\mu}_n$ from μ should be of order $\frac{\sigma}{\sqrt{n}}$.

$$\boxed{|\hat{\mu}_n - \mu| \approx \frac{\sigma}{\sqrt{n}}}$$

Solving Problems with Monte-Carlo Simulation Technique

The Probabilistic Basis for Monte-Carlo: Law of Large Numbers (LLN)

Monte-Carlo Illustration of LLN with Uniform and Exponential Distributions:

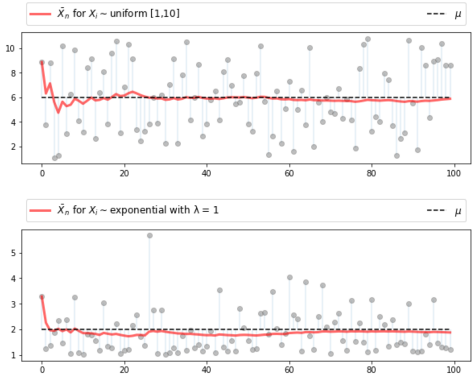


Figure 2: For $n = 100$

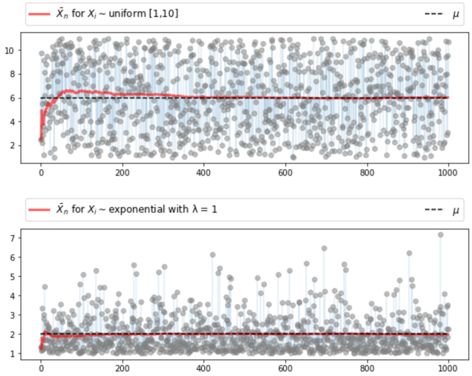


Figure 3: For $n = 1000$

The Probabilistic Basis for Monte-Carlo: Law of Large Numbers (LLN)

Monte-Carlo Illustration of LLN Error Estimation with Uniform and Exponential Distributions:

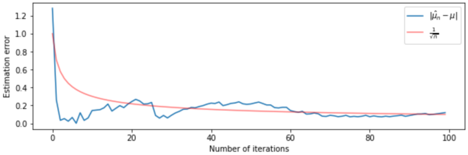


Figure 4: For $n = 100$

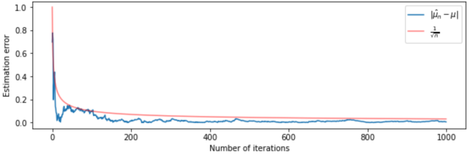


Figure 5: For $n = 1000$

Solving Problems with Monte-Carlo Simulation Technique

The Probabilistic Basis for Monte-Carlo: Central Limit Theorem (CLT)

A more precise statement of how $\hat{\mu}_n$ is distributed can be obtained using: The central limit theorem.

CLT

Let X_n be an independent, and identically distributed (i.i.d.) sequence sampled from any probability distribution $\mathcal{P}$ with mean μ and variance σ^2. Then:

$$\hat{\mu}_n \xrightarrow{n\to\infty} \mathcal{N}\left(\mu, \sigma^2/n\right) \Longleftrightarrow \left(Z = \frac{\hat{\mu}_n - \mu}{\sigma/\sqrt{n}}\right) \xrightarrow{n\to\infty} \mathcal{N}(0,1)$$

Important Notes:

- The distribution of independent sample means is an approximately normal distribution, even if the population is not normally distributed.
- in CLT, given a dataset with an unknown distribution, the sample's means will approximate the normal distribution.

Solving Problems with Monte-Carlo Simulation Technique

The Probabilistic Basis for Monte-Carlo: Confidence Interval

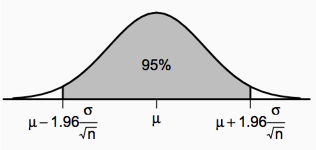
- From Statistic theory, for 95% of the time, $\hat{\mu}_n$ will be within $1.96\frac{\sigma}{\sqrt{n}}$ from μ.
- Alternatively, for 95% of the time, μ will be within $1.96\frac{\sigma}{\sqrt{n}}$ from $\hat{\mu}_n$.
- Hence, we call the interval:

$$\hat{\mu}_n \pm 1.96\frac{\sigma}{\sqrt{n}} = \left[\hat{\mu}_n - 1.96\frac{\sigma}{\sqrt{n}}, \hat{\mu}_n + 1.96\frac{\sigma}{\sqrt{n}}\right]$$

or

$$\mu \pm 1.96\frac{\sigma}{\sqrt{n}} = \left[\mu - 1.96\frac{\sigma}{\sqrt{n}}, \mu + 1.96\frac{\sigma}{\sqrt{n}}\right]$$

a 95% **confidence interval for** μ.



Important Notes: σ can be approximated using the sample standard deviation

Solving Problems with Monte-Carlo Simulation Technique

Central Limit Theorem (CLT): Illustration with Uniform and Exponential Distributions

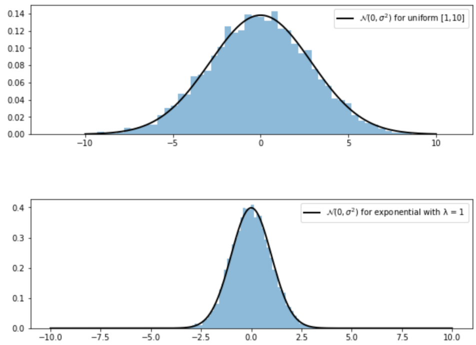


Figure 6: For $n = 100$

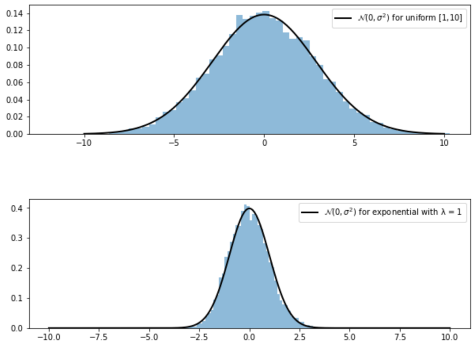


Figure 7: For $n = 1000$

Solving Problems with Monte-Carlo Simulation Technique

Monte Carlo simulation: Basic steps

1. Define **Possible Inputs**
2. **Generate** Inputs Randomly
3. **Computation** on the Inputs
4. **Aggregate the** Results

Define the **domain of possible inputs**.

- The simulated "universe" should be similar to the universe whose behavior we wish to describe and investigate.

Solving Problems with Monte-Carlo Simulation Technique

Monte Carlo simulation: Basic steps

1. Define **Possible Inputs**
2. **Generate** Inputs Randomly
3. **Computation** on the Inputs
4. **Aggregate the** Results

Generate inputs randomly from a probability distribution over the domain.

- inputs should be generated so that their characteristics are similar to the real universe we are trying to simulate
- in particular, dependencies between the inputs should be represented

Solving Problems with Monte-Carlo Simulation Technique

Monte Carlo simulation: Basic steps

1. Define **Possible Inputs**
2. **Generate** Inputs Randomly
3. **Computation** on the Inputs
4. **Aggregate the** Results

The **computation** should be deterministic.

Solving Problems with Monte-Carlo Simulation Technique

Monte Carlo simulation: Basic steps

1. Define **Possible Inputs**
2. **Generate** Inputs Randomly
3. **Computation** on the Inputs
4. **Aggregate the** Results

Aggregate the results to obtain the output of interest.

Typical outputs:

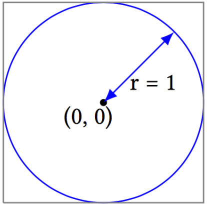
- histogram
- summary statistics(mean,standard deviation...)
- confidence intervals,
- ...

Solving Problems with Monte-Carlo Simulation Technique

Monte Carlo simulation: Basic steps illustration Example

Example: π Estimation

- Consider the largest circle which can be fit in the square ranging on $\mathbb{R}^2$ over $[-1, 1]^2$.
  - the circle has radius 1 and area π.
  - the square has an area of $2^2 = 4$.
  - The ratio between their areas is thus $\frac{\pi}{4}$
- π can be approximated using the following Monte-Carlo procedure:
  1. draw the square over$[-1, 1]^2$.
  2. draw the largest circle that fits inside the square.
  3. randomly scatter a large number$N$ of points over the square.
  4. count how many point fell inside the circle.
  5. the count divided by$N$ and multiplied by 4 is an approximation of π.



Solving Problems with Monte-Carlo Simulation Technique

Monte Carlo simulation: Basic steps illustration Example

Example: π Estimation

1. Define **Possible Inputs**
2. **Generate** Inputs Randomly
3. **Computation** on the Inputs
4. **Aggregate the** Results

All points within the $[-1,1]^2$ unit square, uniformly distributed.

Monte Carlo simulation: Basic steps illustration Example

Example: π Estimation

1. Define **Possible Inputs**
2. **Generate** Inputs Randomly
3. **Computation** on the Inputs
4. **Aggregate the** Results

Generate N uniform random points of (x,y) from the unit square in Python:

```
N = 10000
points = numpy.random.uniform(-1, 1, size=(N,2))
```

Monte Carlo simulation: Basic steps illustration Example

Example: π Estimation

1. Define **Possible Inputs**
2. **Generate** Inputs Randomly
3. **Computation** on the Inputs
4. **Aggregate the** Results

Test whether a randomly generated point(x, y) is within the circle:

```
for point in points:
    x = point[0]
    y = point[1]

     if (np.sqrt(x**2 + y**2)) < 1:
        print("The point is inside")
    else:
        print("The point is outside")
```

Solving Problems with Monte-Carlo Simulation Technique

Monte Carlo simulation: Basic steps illustration Example

Example: π Estimation

1. Define **Possible Inputs**
2. **Generate** Inputs Randomly
3. **Computation** on the Inputs
4. **Aggregate the** Results

Count the proportion of points that are within the circle:

```python
N = 10000
inside = 0
outside = 0
for point in points:
    x,y = point[0],point[1]
    if (np.sqrt(x**2 + y**2)) < 1:
        inside += 1
    else:
        outside += 1
pi = 4*inside / float(N)
print("Estimated Pi: {}".format(pi))
```

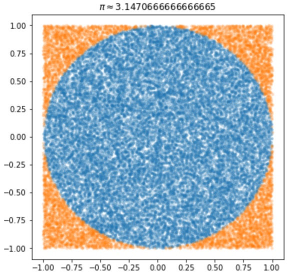
Solving Problems with Monte-Carlo Simulation Technique

Monte Carlo simulation: Basic steps illustration Example

Example: π Estimation

Putting it all together: here is an implementation in Python with the NumPy library.

```
import numpy as np
np.random.seed(42)
N = 100000
inside,outside = [],[]
points = np.random.uniform(-1, 1, size=(N
    ,2))
for point in random_points:
    x = point[0]
    y = point[1]
    if (np.sqrt(x**2 + y**2)) < 1:
        inside.append((x, y))
    else:
        outside.append((x, y))
pi = 4*len(inside)/ len(random_points)
```



Solving Problems with Monte-Carlo Simulation Technique

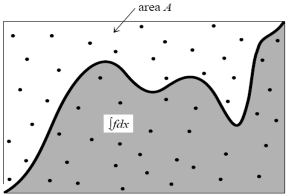
Monte-Carlo Applications in Resolving Numerical Integrals

- Assume we want to evaluate an integral:

$$\int_I f(x)dx$$

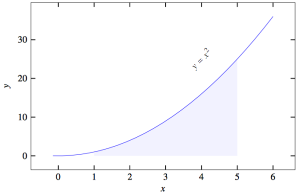
- **Principle:** The integral to compute is related to the expectation of a random variable

$$\mathbb{E}(f(x)) = \int_I f(x)dx$$



- **Method:**
  - Sample points within I
  - Calculate the mean of the random variable within I
  - Integral $=$ sampled area $\times$ mean

Monte-Carlo Applications in Resolving Numerical Integrals: Integration Example



Task: find the shaded area, $\int_1^5 x^2 dx$

Analytical solution:

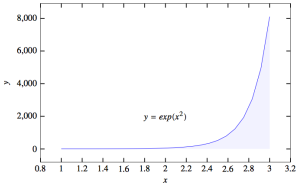
```
import sympy as sy
x = sy.Symbol("x")
i = sy.integrate(x**2)
i.subs(x, 5) - i.subs(x, 1)
124/3
float(i.subs(x, 5) - i.subs(
    x, 1))
41.333333333333336
```

Numerical solution:

```
import numpy as np
N = 100000
accum = 0
for i in range(N):
    x = np.random.uniform(1,
    5)
    accum += x**2
area = 5-1
integral = area*accum/float(
   N)
41.282652648597875
```

Solving Problems with Monte-Carlo Simulation Technique

Monte-Carlo Applications in Resolving Numerical Integrals: Integration Example



Task: find the shaded area, $\int_1^3 e^{x^2} dx$

Analytical solution:

```
import sympy as sy
x = sy.Symbol("x")
i = sy.integrate(sy.exp(x
    **2))
i.subs(x, 3) - i.subs(x, 1)
float(i.subs(x, 3) - i.subs(
    x, 1))
1443.08247146807
```

Numerical solution:

```
import numpy as np
N = 100000
accum = 0
for i in range(N):
    x = np.random.uniform(1,
    3)
    accum += np.exp(x**2)
area = 3 - 1
integral = area * accum /
    float(N)
1441.9954276924555
```

Solving Problems with Monte-Carlo Simulation Technique

Monte-Carlo Applications in Resolving Numerical Integrals: 2D integration example

Task: resolve the double integral, $\int_0^1 \int_4^6 cos(x^4) + 3y^2 dxdy$

Analytical solution:

```
import sympy as sy
x = sy.Symbol("x")
y = sy.Symbol("y")
d1 = sy.integrate(sy.cos(x**4) + 3 * y**2, x)
d2 = sy.integrate(d1.subs(x, 6) - d1.subs(x, 4), y)
sol = d2.subs(y, 1) - d2.subs(y, 0)
float(sol)
2.00505508674967 4
```

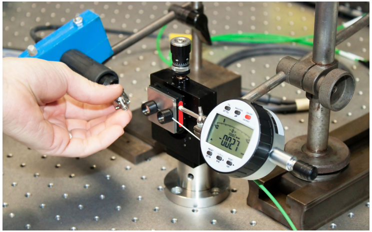
Numerical solution:

```
import numpy as np
N = 100000
accum = 0
for i in range(N):
    x = np.random.uniform(4, 6)
    y = np.random.uniform(0, 1)
    accum += np.cos(x**4) + 3 * y * y
volume = (6-4) * (1-0)
integral = volume * accum/float(N)
2.0070240947478672
```

Monte-Carlo Applications to Uncertainty Analysis

- **Uncertainty Analysis:** investigates the uncertainty of variables that are used in decision-making problems.
- **Uncertainty** is inherent in everything we do.

In physical experiment



In numerical experiments

- **Our goal:**
  - Quantify uncertainties propagated in models variables.
  - Understand and model it to make better decisions

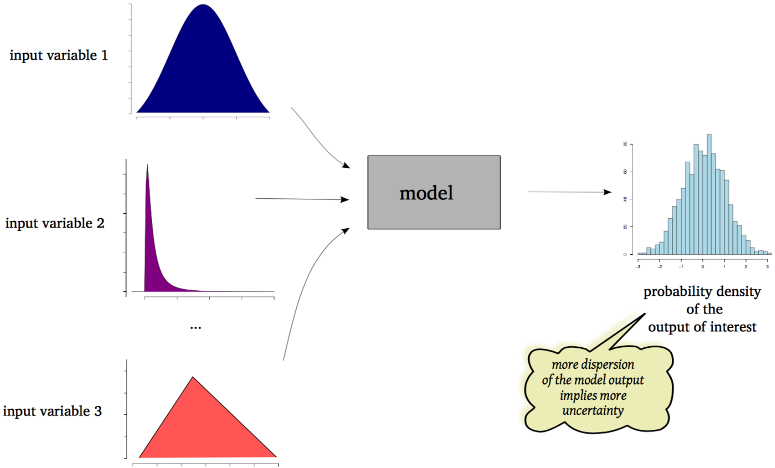
Solving Problems with Monte-Carlo Simulation Technique

Monte-Carlo Applications to Uncertainty Analysis: Basics Steps

- **Uncertainty analysis:** propagate uncertainty on input variables through the model to obtain a probability distribution of the output(s) of interest
- **Principle:** Uncertainty on each input variable is characterized by a probability density function
- Run the model a large number of times with input values drawn randomly from their PDF.
- Aggregate the output uncertainty as a probability distribution

Solving Problems with Monte-Carlo Simulation Technique

Monte-Carlo Applications to Uncertainty Analysis: Basics Steps



Solving Problems with Monte-Carlo Simulation Technique

Monte-Carlo Applications to Uncertainty Analysis: Example for uncertainty propagation

- Let X be a measure uniformly distributed over $[0, 100]$ and Y another measure exponentially distributed with $\beta = 2.25$.
- We are interested in the distribution of the output variable:

$$Z = X * Y$$

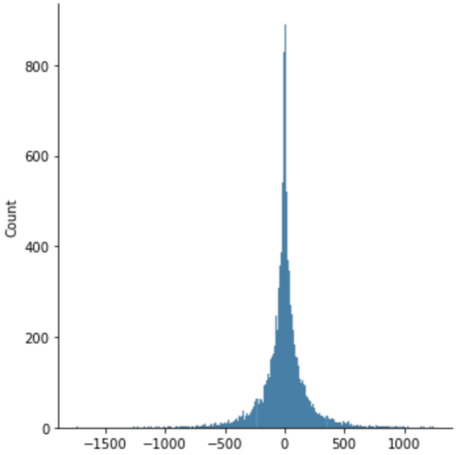
Q: What is the 95th percentile of Z?

Numerical solution:

```
import  numpy as np
N = 10000
Z = np.zeros(N)
for i in range(N):
    x = np.random.uniform(-100, 100)
    y = np.random.exponential(2.25)
    Z[i] = x * y
print(np.percentile(zs, 95))
sns.displot(zs)
277.2387365659431 6
```

Solving Problems with Monte-Carlo Simulation Technique

Monte-Carlo Applications to Uncertainty Analysis: Example for uncertainty propagation



End of Lecture 5:

Solving Problems with Monte-Carlo Simulation Technique