Étienne Long, Xixuan Li
Probability and Statistics
Professor François Charette
22 March 2024

Monte Carlo Simulation

Étienne Long, Xixuan Li, Marianopolis College, Montreal, Quebec, Canada.

Contents

1 Introduction

The research paper explores the application of Monte Carlo simulation, a computational method employed for the evaluation or approximation of definite integrals that challenge direct integration. It entails the generation of random points within the interval bounds, applying the function f, and computing the area of the integral as $f(x) \times (b - a)$. Subsequently, the average of these computed areas is calculated to yield an approximation of the integral value. The research's theoretical foundation is based on an examination of Chapter 24.2 in Larry Wasserman's "All of Statistics". Practical implementation is demonstrated using Python supplemented with NumPy to illustrate examples drawn from the textbook. Furthermore, the simulation is applied to estimate the value of π by evaluating the integral of the function $\sqrt{x^2 - 1}$ over the interval 0 to 1, representing the area of a semicircle with a 1-unit radius, and subsequently multiplying this computed area by 4.

2 Example 24.1

For the example 24.1, we approximate the value of the definite integral $h(x) = x^3$ by randomly selecting N points within the bounds. We evaluate the function $h(x)$ at each point and compute the area of the integral as $f(x) \times (b - a)$, where $f(x)$ is the function value and $b - a$ is the interval width. Finally, we calculate the average of these areas.

1. Generate N points from the function $h(x) = x^3$ from a to b.

```
import numpy as np
import random
import sympy as sy #used to find the exact integral value

#define a, b, N and h(x)
a=0
b=1
N=100
def h(x):
  return x**3
```

2. Find

$$\hat{I} = \frac{1}{N} \sum_{i=1}^{N} w(X_i)$$

where $w(X_i) = X_i(b-a)$

```
sum=0
for i in range(N):
  sum+=h(random.uniform(a,b))
sum = (b-a)*sum
print(sum)
Ihat=1/N*sum
print(Ihat)
```

We get $\hat{\text{I}} = 0.2417854893547572$. Next, we calculate for the percentage error.

```
x = sy.Symbol("x")
integralvalue=sy.integrate(h(x),(x,a,b))
print(integralvalue)
percenterror = abs((Ihat-integralvalue)*100/integralvalue)
print(Percenterror,"%")
```

Given that the precise integral of x^3 from 0 to 1 is 0.25 and our Monte Carlo simulation yields a percentage error of 3.28580425809712%, it demonstrates a reasonably close approximation to the true value. It's important to recognize that due to the random selection of points in a uniform distribution, each trial will produce a different sum. Consequently, the percentage error may not consistently remain at 1.61%.

3. Explore the outcomes when altering the value of N. Repeat these procedures for $N = 1000$, 10000, 100000, and 1000000.

```
N=10
for i in range(5):
  N*=10
  print("N is:",N)
  sum=0
  for i in range(N):
    sum=sum+h(random.uniform(a,b))
  sum = (b-a)*sum
  Ihat=1/N*sum
  print("I hat is:",Ihat)
  percenterror = abs((Ihat-integralvalue)*100/integralvalue)
```

```
  print("Percentage error :",percenterror,"%")
  print()
```

We obtain the following results:

- For $N = 100$:
  - Estimated integral ($\hat{I}$): 0.23644516706604016
  - Percentage error: 5.42193317358394%
- For $N = 1000$:
  - Estimated integral ($\hat{I}$): 0.24527452206910852
  - Percentage error: 1.89019117235659%
- For $N = 10000$:
  - Estimated integral ($\hat{I}$): 0.24596707185253205
  - Percentage error: 1.61317125898718%
- For $N = 100000$:
  - Estimated integral ($\hat{I}$): 0.24868493708934788
  - Percentage error: 0.526025164260846%
- For $N = 1000000$:
  - Estimated integral ($\hat{I}$): 0.2499858421952935
  - Percentage error: 0.00566312188260421%

Hence, we observe that as the sample size grows, the estimation becomes increasingly precise, approaching the exact value of 0.25.

4. Switch up the value of b for experimentation while maintaining N at 1000000.

```
N=1000000
b=0
for i in range(5):
  b+=1
  sum=0
  for i in range(N):
    sum+=h(random.uniform(a,b))
  sum = (b-a)*sum
  Ihat=1/N*sum
  print("I hat from",a,"to",b,"is:",Ihat)
  integralvalue=sy.integrate(h(x),(x,a,b))
  print("actual integral value is:",integralvalue)
  percenterror = abs((Ihat-integralvalue)*100/integralvalue)
  print("Percentage error:",percenterror,"%")
  print()
```

We obtain the following results:

- For x from 0 to 1:
  - Estimated integral ($\hat{I}$): 0.25005418143917824
  - Actual integral value: $\frac{1}{4}$
  - Percentage error: 0.0216725756712943%
- For x from 0 to 2:
  - Estimated integral ($\hat{I}$): 3.994489353952493
  - Actual integral value: 4
  - Percentage error: 0.137766151187679%
- For x from 0 to 3:
  - Estimated integral ($\hat{I}$): 20.23748653417094
  - Actual integral value: $\frac{81}{4}$
  - Percentage error: 0.0617948929830079%
- For x from 0 to 4:
  - Estimated integral ($\hat{I}$): 63.935650182073815
  - Actual integral value: 64
  - Percentage error: 0.100546590509665%
- For x from 0 to 5:
  - Estimated integral ($\hat{I}$): 156.19112166907306
  - Actual integral value: $\frac{625}{4}$
  - Percentage error: 0.0376821317932445%

Observe the variation in percentage error as it correlates with the bounds of the integral. Although the difference may seem insignificant, it's worth noting since we're dealing with an extensive number of random points within a narrow range of interval differences.

3 Example 24.2

The objective is to approximate the value of the cumulative distribution function (CDF) of the standard normal distribution at a specific point s. This is achieved by employing a method wherein N random points are sampled from the standard normal probability density function (PDF). For each sampled point that is smaller than s, a counter is incremented by 1. Eventually, the total count is divided by N to yield an estimation of the CDF value at s. This process, which can also be expressed as (number of observations $\leq s$) $\times \frac{1}{N}$, provides a practical means of approximating the desired CDF value.

1. Generate N points from the standard normal distribution $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ and define N and s. We also define a function $h(x)$ such that $h(x) = 1$ If $s < x$, then $h(x) = 0$ if $x \geq s$.

```
from scipy.stats import norm
N=100
s=1
def h2(x):
  if x<s:
    return 1
  if x>=s:
    return 0
```

3. Find

$$\hat{I} = \frac{1}{N} \sum_i h(X_i)$$

and compare with the exact value of Φ

```
sum=0
N=100
for i in range(N):
  sum+=h2(np.random.standard_normal())

Ihat2=1/N*sum
print("Estimated Phi:", Ihat2)
print("Exact Phi", norm.cdf(s))
percenterror2=abs((Ihat2-norm.cdf(s))*100/norm.cdf(s))
print("Percentage error:", percenterror2,"%")
```

We obtain the following results:

- Estimated Φ: 0.86
- Exact Φ: 0.8413447460685429
- Percentage error: 2.217313891675179%

4. Explore the effects of varying the value of N and repeat the steps for N ranging from 100 to $1,000,000$.

```
N=10
print("Exact Phi:", norm.cdf(s))
print()
for i in range(5):
  N*=10
  sum=0
  print("N is:",N)
  for j in range(N):
    sum+=h2(np.random.standard_normal())
  Ihat2=1/N*sum
  print("Estimated Phi:", Ihat2)
  percenterror2=abs((Ihat2-norm.cdf(s))*100/norm.cdf(s))
  print("Percentage error:",percenterror2,"%")
  print()
```

We obtain the following results:

- Exact Φ: 0.8413447460685429
- For $N = 100$:
  - Estimated Φ: 0.8300000000000001
  - Percentage error: 1.3484063603599918%
- For $N = 1000$:
  - Estimated Φ: 0.833
  - Percentage error: 0.9918343351564866%
- For $N = 10000$:
  - Estimated Φ: 0.8385
  - Percentage error: 0.3381189556166963%
- For $N = 100000$:
  - Estimated Φ: 0.84267
  - Percentage error: 0.15751615941619468%
- For $N = 1000000$:
  - Estimated Φ: 0.842063
  - Percentage error: 0.08536975298334665%

5. Experiment with different values of s, maintaining $N = 1000000$.

```
N=1000000
s=-3

print()
for i in range(5):
  s+=1
  sum=0
  print("s is:",s)
  print("Exact Phi:", norm.cdf(s))
  for j in range(N):
    sum+=h2(np.random.standard_normal())
  Ihat2=1/N*sum
  print("Estimated Phi:", Ihat2)
  percenterror2=abs((Ihat2-norm.cdf(s))*100/norm.cdf(s))
  print("Percentage error is:",percenterror2,"%")
  print()
```

We obtain the following results:

- $s = -2$:
  - Exact Φ: 0.022750131948179195
  - Estimated Φ: 0.022421

- Percentage error: 1.4467254472585018%

- $s = -1$:
  - Exact Φ: 0.15865525393145707
  - Estimated Φ: 0.15791
  - Percentage error: 0.46973164328932165%

- $s = 0$:
  - Exact Φ: 0.5
  - Estimated Φ: 0.5007159999999999
  - Percentage error: 0.14319999999998778%

- $s = 1$:
  - Exact Φ: 0.8413447460685429
  - Estimated Φ: 0.841375
  - Percentage error: 0.0035959018700038926%

- $s = 2$:
  - Exact Φ: 0.9772498680518208
  - Estimated Φ: 0.977294
  - Percentage error: 0.004515932886968109 5%

4 π Estimation

The goal is to estimate the value of π. This is achieved by determining the area of a quarter circle with a radius of 1 and subsequently multiplying the result by 4.

1. Define $h(x) = \sqrt{x^2 - 1}$.

```
def h3(x):
  return (1-x**2)**0.5
```

2. Repeat steps from Example 24.1.

```
a=0
b=1
N=10
for i in range(5):
  N*=10
  print("N is:",N)
  sum=0
  for i in range(N):
    sum+=h3(random.uniform(a,b))
  sum = (b-a)*sum
  Ihat3=1/N*sum
  print("Estimated value of pi is:",4*Ihat3)
```

```
percenterror3=abs((np.pi-4*Ihat3)*100/np.pi)
print("Percentage error is:", percenterror3,"%")
print()
```

We obtain the following results:

- For $N = 100$:
  - Estimated value of π: 3.0260493064940888
  - Percentage error: 3.677858966332787%
- For $N = 1000$:
  - Estimated value of π: 3.10650850718489
  - Percentage error: 1.1167630649000189%
- For $N = 10000$:
  - Estimated value of π: 3.1617304493320293
  - Percentage error: 0.6410059470703622%
- For $N = 100000$:
  - Estimated value of π: 3.1424282426756522
  - Percentage error: 0.026597626681623517%
- For $N = 1000000$:
  - Estimated value of π: 3.1416425386144136
  - Percentage error: 0.0015878896509209154%

5 Conclusion

In summary, Monte Carlo simulations prove effective when N reaches a significant magnitude. With N exceeding 10,000, the percentage errors consistently remained below 1% across all three scenarios. However, counter-intuitive outcomes occasionally arise. Increasing N by a factor of 10 may paradoxically result in larger percentage errors. This phenomenon occurs because while higher N increases the likelihood of converging to the actual value, randomness still plays a significant role.