Quasi-Monte Carlo Methods

Quasi-Monte Carlo Simulations

Quasi-Monte Carlo simulations are variations of Monte Carlo simulations.

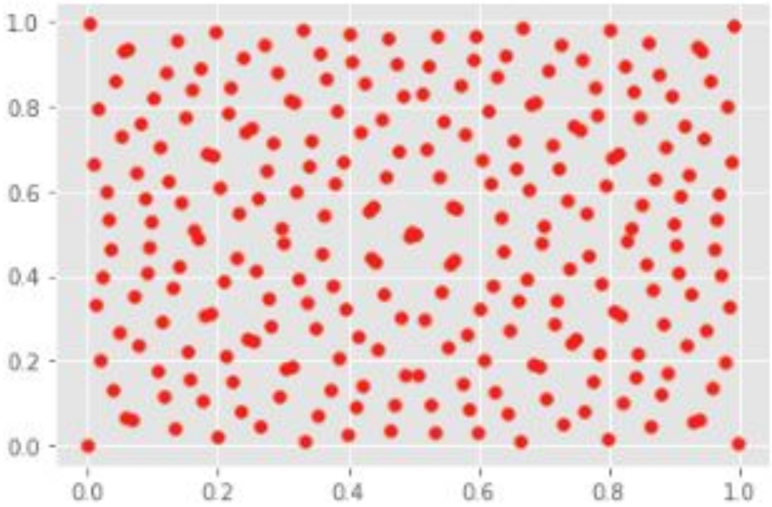
It uses **low-discrepancy deterministic sequences** with **better uniformity** properties than purely random sequences providing **more accurate and efficient estimates** compared to traditional Monte Carlo simulations.

Example: **Halton** or **Sobol** sequences.

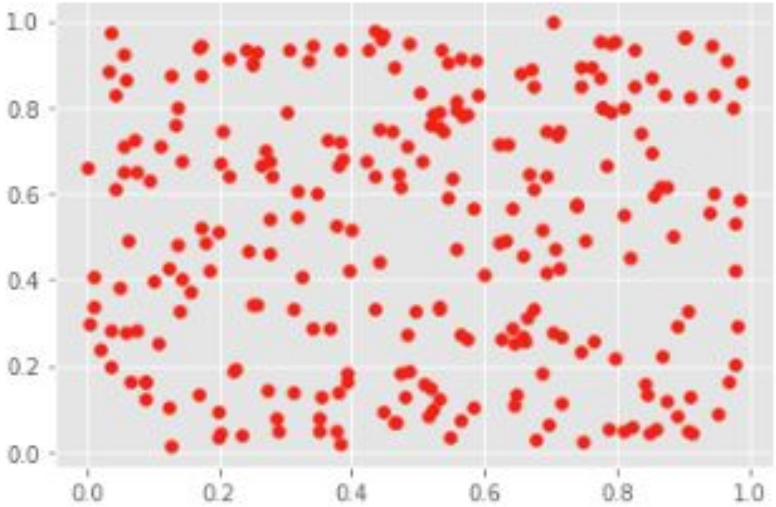
Monte Carlo vs Quasi-Monte Carlo Simulations

We generate below 256 points uniformly distributed in $[0,1]^2$ with Monte Carlo and Quasi-Monte Carlo (Sobol sequence) methods.

Quasi-Monte Carlo



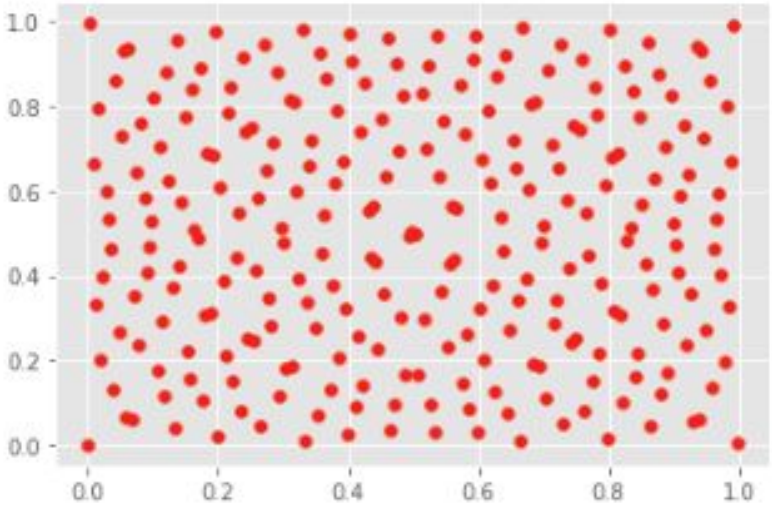
Monte Carlo



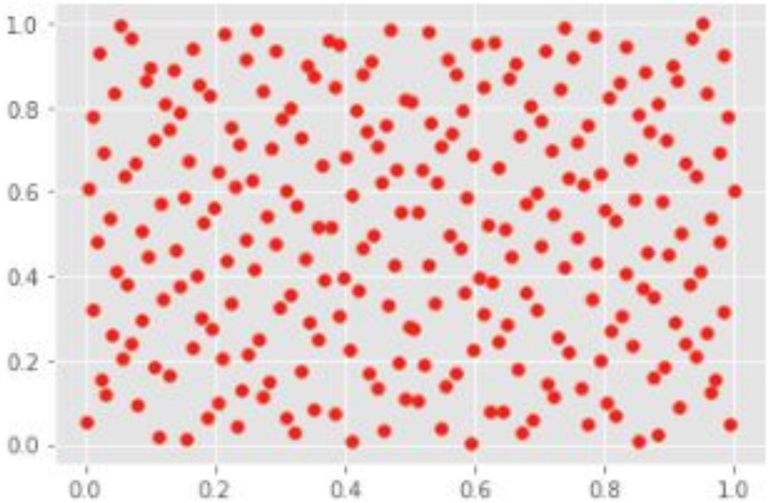
Monte Carlo vs Quasi-Monte Carlo Simulations

We generate below 256 points uniformly distributed in $[0,1]^2$ with Monte Carlo and Quasi-Monte Carlo methods.

Quasi-Monte Carlo



Quasi-Monte Carlo with Scrambling



Monte Carlo vs Quasi-Monte Carlo Simulations

Quasi-Monte Carlo allows more accurate and efficient estimates with a rate of convergence close to $O(\log(N)^k N^{-1})$ for a problem of dimension k compared to $O(N^{-0.5})$ with Monte Carlo simulations.

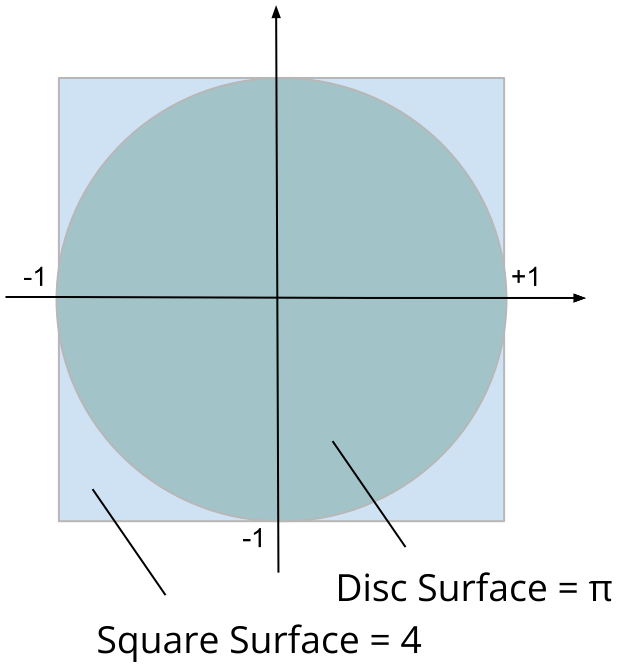
We would get roughly the same accuracy of the estimate with 10^4 quasi-Monte Carlo simulations than with 10^7 Monte Carlo simulations for a problem of dimension 1!!

Monte Carlo vs Quasi-Monte Carlo Simulations: Pi Estimation

```
import numpy as np
from scipy.stats import qmc
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

```
def MC_Pi(n):
    x1=np.random.uniform(-1,1,n)
    x2=np.random.uniform(-1,1,n)

    y = x1[np.sqrt(x1**2+x2**2)<1]
    pi_est = len(y) / n * 4

    return pi_est
```

```
def QMC_Pi(n):

    sobol = qmc.Sobol(d = 2, scramble = True) #Sobol sequence
    x = sobol.random(2**n) # Generate Quasi-Monte Carlo random numbers

    x1 = x[:,0]
    x2 = x[:,1]

    y = x1[np.sqrt(x1**2+x2**2)<1]
    pi_est = len(y) / 2**n * 4

    return pi_est
```



Monte Carlo vs Quasi-Monte Carlo Simulations: Pi Estimation

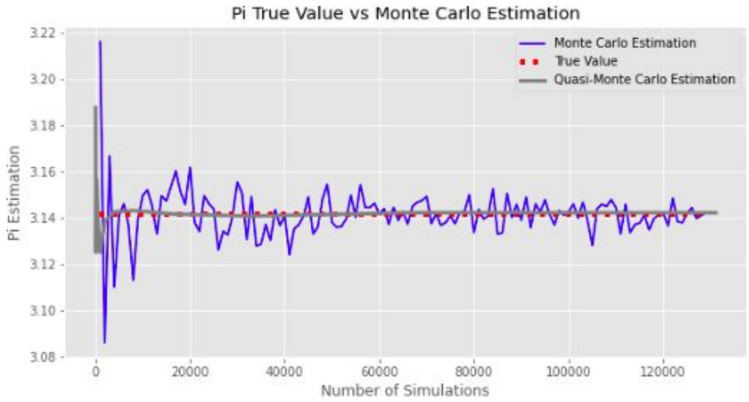
```
import numpy as np
from scipy.stats import qmc
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

```
def MC_Pi(n):
    x1=np.random.uniform(-1,1,n)
    x2=np.random.uniform(-1,1,n)

    y = x1[np.sqrt(x1**2+x2**2)<1]
    pi_est = len(y) / n * 4

    return pi_est
```

```
def QMC_Pi(n):

    sobol = qmc.Sobol(d = 2, scramble = True) #Sobol sequence
    x = sobol.random(2**n) # Generate Quasi-Monte Carlo random numbers

    x1 = x[:,0]
    x2 = x[:,1]

    y = x1[np.sqrt(x1**2+x2**2)<1]
    pi_est = len(y) / 2**n * 4

    return pi_est
```

```
Pi_MC = MC_Pi(2**20)
Pi_QMC = QMC_Pi(20)
print("Pi MC Estimation: " + str(np.round(Pi_MC, 6)))
print("Pi QMC Estimation: " + str(np.round(Pi_QMC, 6)))
print("Pi True Value: " + str(np.round(np.pi, 6)))
```

```
Pi MC Estimation: 3.140198
Pi QMC Estimation: 3.141491
Pi True Value: 3.141593
```



Contact Us

website: **www.quant-next.com**

email: **contact@quant-next.com**

Follow us on LinkedIn

Disclaimer

This document is for educational and information purposes only.

It does not intend to be and does not constitute financial advice, investment advice, trading advice or any other advice, recommendation or promotion of any particular investments.