Monte Carlo Simulation

Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying idea is the weak law of large numbers.

> Let $X_1, X_2, \ldots$ be i.i.d. with $E|X_i| < \infty$. Let $S_n = X_1 + \cdots + X_n$ and let $\mu = EX_1$. Then $S_n/n \to \mu$ in probability.

Loosely speaking, **the expectation** can be well approximated by **the sample analogue** in probability as sample size n goes to infinity.

We will apply the monte carlo simulation to compute the expectations of interest.

- $Eh(X) = \int h(x)f(x)dx$ is well approximated by $\frac{1}{n}\sum_{i=1}^n h(X_i)$ as n goes to infinity, where $X_1, \ldots, X_n$ are i.i.d. with common pdf $f(x)$.
- Similarly, $Pr(X \leq x)$ is well approximated by $\frac{1}{n}\sum_{i=1}^n 1\{X_i \leq x\}$ as n goes to infinity.
  - Probabilities can be expressed as **expectations of indicator functions**, i.e.,

$$Pr(X \leq x) = \int_{-\infty}^{x} f(t)dt = \int_{-\infty}^{\infty} 1_{(-\infty,x]}(t)f(t)dt = E1_{(-\infty,x]}.$$

Example: $Pr(4 < X \leq 5)$

Consider the random variable $X \sim N(1, 4)$. We are interested in $Pr(4 \leq X < 5)$.

The `norm.cdf()` function can be used to compute the target probability directly.

In [1]:

```python
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt
mu = 1
sigma = 2
print("The probability that 4 < X <= 5 is %.3f" % (norm.cdf(5, mu, sigma) - norm
```

```
The probability that 4 < X <= 5 is 0.044
```

We can also get the approximate probability through Monte Carlo simulation.

First, generate 5000 random draws from the target distribution, i.e., $X_1, X_2, \ldots, X_{5000} \sim N(1, 4)$.

In [2]:

```python
np.random.seed(123) # for reproducible randomness
nsim = 5000
```

```
X = norm.rvs(loc = mu, scale = sigma, size = nsim)
print("The first 5 random draws are ", X[:5])
```

```
The first 5 random draws are  [-1.17126121  2.99469089  1.565957   -2.01258943 -
0.1572005 ]
```

In [3]:

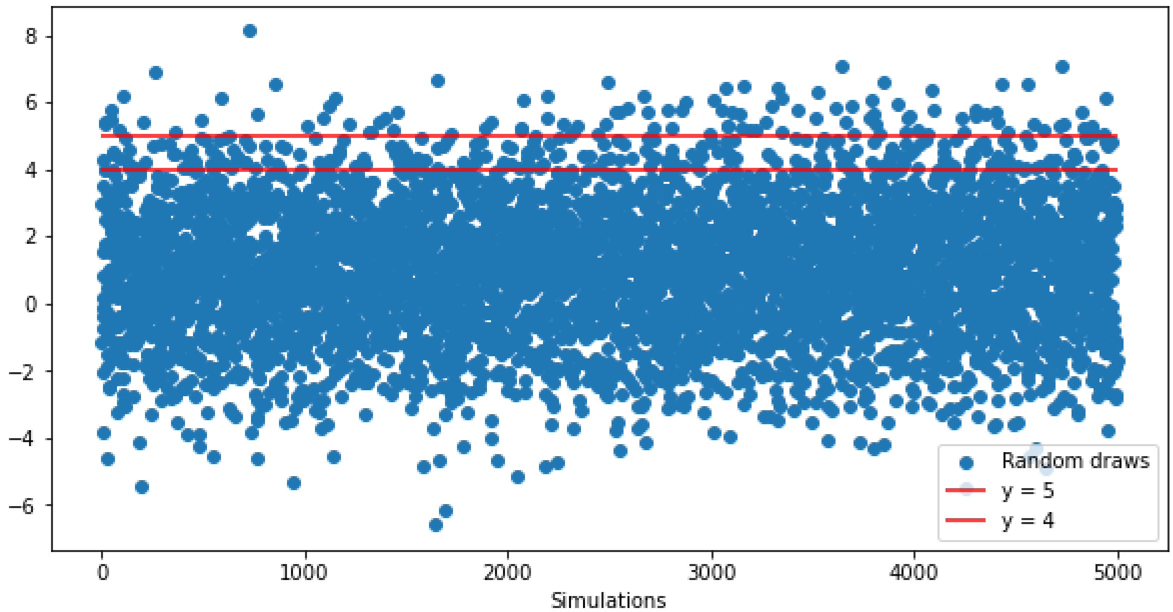
```
np.random.seed(123) # given the same seed, we have the same random draws
nsim = 5000
norm.rvs(loc = mu, scale = sigma, size = nsim)[:5]
```

Out[3]:

```
array([-1.17126121,  2.99469089,  1.565957  , -2.01258943, -0.1572005 ])
```

In [4]:

```
plt.figure(figsize = (10, 5))
plt.scatter(np.linspace(1, nsim, nsim), X, label = 'Random draws')
plt.hlines(y = 5, xmin = 1, xmax = nsim, color = 'red', label = 'y = 5')
plt.hlines(y = 4, xmin = 1, xmax = nsim, color = 'red', label = 'y = 4')
plt.xlabel('Simulations')
plt.legend()
plt.show()
```



Now we compute the fraction of realizations $(X_1, \ldots, X_{5000})$ falling between 4 and 5. This fraction is "approximately" the probability of interest when the number of runs is very large. (Weak law of large numbers)

In [5]:

```
Y = (X > 4) * (X <= 5) * 1 # 5000 indicator variables
print("The first five results: ", Y[:5])
print("The fraction of X's satisfying the condition is %.3f" % (np.mean(Y) ))
```

```
The first five results:  [0 0 0 0 0]
The fraction of X's satisfying the condition is 0.045
```

Note that 0.045 is quite close to the true probability 0.044 as computed above through `norm.cdf()` .

Example: $P(T \leq 0.6)$, where $T \sim t(5)$

Similarly, we use Monte Carlo simulation to conveniently approximate $P(T \leq 0.6)$, where $T \sim t(5)$.

In [6]:

```
from scipy.stats import t
x0 = 0.6
print('The probability that T <= {} is {:.3f}'.format(x0, t.cdf(x0, df = 5)))

np.random.seed(123)
nsim = 5000
X = t.rvs(df = 5, size = nsim)
print("The fraction of X's at most {} is {:.3f}".format(x0, np.mean(X <= x0)))
```

```
The probability that T <= 0.6 is 0.713
The fraction of X's at most 0.6 is 0.707
```

Example: $Pr(-1 \leq X_1 < 1, -2 \leq X_2 < 3)$

Suppose that

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 \\ 0.5 & 4 \end{pmatrix}\right).$$

We can apply the monte carlo simulation to computing joint probabilities.

In [7]:

```
from scipy.stats import multivariate_normal
import pandas as pd
mu = np.array([0, 0])
cov = np.array([[1, 0.5], [0.5, 4]])
diff = multivariate_normal.cdf(np.array([1,3]), mean = mu, cov = cov) \
    - multivariate_normal.cdf(np.array([-1,3]), mean = mu, cov = cov) \
    - multivariate_normal.cdf(np.array([1,-2]), mean = mu, cov = cov) \
    + multivariate_normal.cdf(np.array([-1,-2]), mean = mu, cov = cov)
print("The target probability:", diff)


np.random.seed(123)
nsim = 50000
X = multivariate_normal.rvs(mean=mu, cov=cov, size=nsim)
result = (X[:, 0] < 1) * (X[:, 0] >= -1) * (X[:, 1] < 3) * (X[:, 1] >= -2) * 1
print("The fraction of random draws satisfying the condition is %.3f" % (np.sum(
```

```
The target probability: 0.5354662279197304
The fraction of random draws satisfying the condition is 0.535
```

In [8]:

```
result[:5]
```

Out[8]:

```
array([0, 0, 0, 0, 1])
```

Example: $Pr(-3 \leq 2X_1 + 3X_2 < 3)$

Next we consider a new random variable $Y \equiv 2X_1 + 3X_2$. What is the probability $Pr(-3 \leq Y < 3)$?

Note that Y is normally distributed with mean 0 and variance $2^2 * Var(X_1) + 3^2 * Var(X_2) + 2 * 2 * 3 * Cov(X_1, X_2) = 46$. So the true probability is

In [9]:

```
print("The true probability is %.3f" % (norm.cdf(3, 0, np.sqrt(46)) - norm.cdf(-
```

```
The true probability is 0.342
```

We can also approximate the probability through Monte Carlo simulation.

In [10]:

```
newvar = 2*X[:, 0] + 3*X[:,1]
fraction = np.mean((newvar >= -3) * (newvar < 3))
print("The fraction of random draws satisfying the condition is %.3f" % fraction
```

```
The fraction of random draws satisfying the condition is 0.342
```

In [11]:

```
newvar
```

Out[11]:

```
array([-5.76922309, -0.28063664, -1.47137681, ...,  0.98165658,
        7.8461253 , 13.25262412])
```

As we can see, the result from the monte carlo simulation is "close to" the true probability. In addition, **you do not need to derive the marginal distribution of the new random variable Y in the MC simulation**.

Example: $E[2X_1 + 3X_2]$ and $Var(2X_1 + 3X_2)$

As derived above, the mean and variance of $2X_1 + 3X_2$ are 0 and 46, repectively. This can be approximated through sample means and sample variance.

In [12]:

```
print('The sample mean is %.3f' % np.mean(newvar))
print("The sample variance is %.3f" % np.var(newvar, ddof = 1))
```

```
The sample mean is 0.013
The sample variance is 46.313
```

Example: Approximating π

Suppose that $X_1, X_2 \overset{i.i.d.}{\sim} Uniform([-1, 1])$. We can draw a lot of samples from this joint distribution. As will be seen shortly, the area of the circle of radius 1 is π while the area of the square $[-1, 1] \times [-1, 1]$ is 4. Roughly, the fraction of the samples falling into the circle is $\pi/4$. Thus, we can infer the value of π through 4 times the fraction.

In [13]:

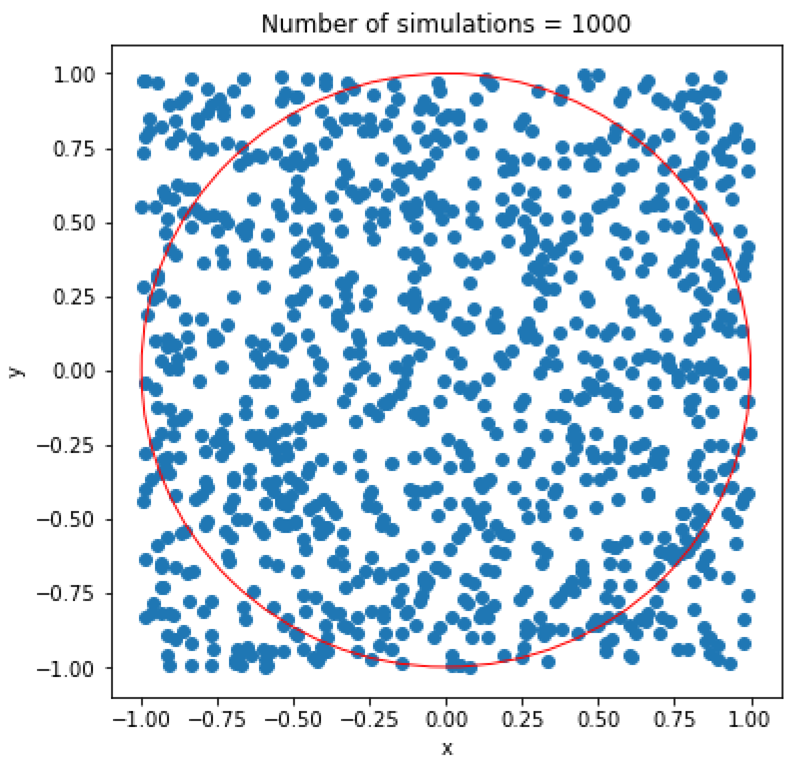
```
np.pi
```

Out[13]:

```
3.141592653589793
```

In [14]:

```
from scipy.stats import uniform

nsim = 1000
X = uniform.rvs(loc = -1, scale = 2, size = (nsim, 2)) #  the uniform distributi

fig = plt.figure(figsize = (6,6))
ax = fig.add_subplot(111)
plt.scatter(X[:, 0], X[:, 1])
circle = plt.Circle((0, 0), 1, color='r', fill=False)
ax.add_patch(circle)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Number of simulations = {}'.format(nsim))
plt.show()

fraction = np.sum(X[:,0] ** 2 + X[:, 1] ** 2 <= 1) / nsim
print("The fraction of samples falling in the circle is %.3f" % fraction)
print("The estimated pi is %.3f" % (4*fraction))
```



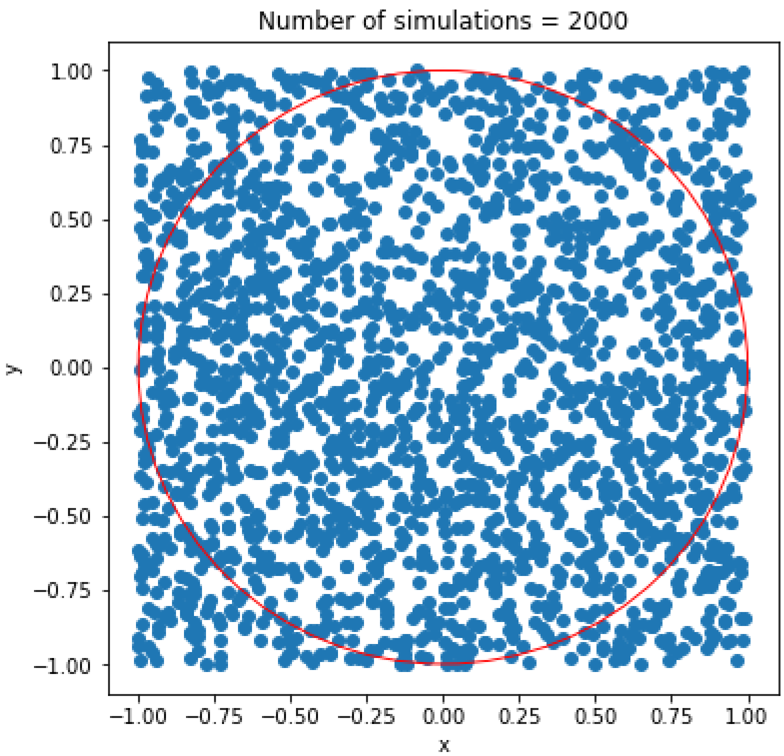

```
The fraction of samples falling in the circle is 0.787
The estimated pi is 3.148
```

We can define a function of `nsim` and see the approximation errors in general decline with `nsim`.

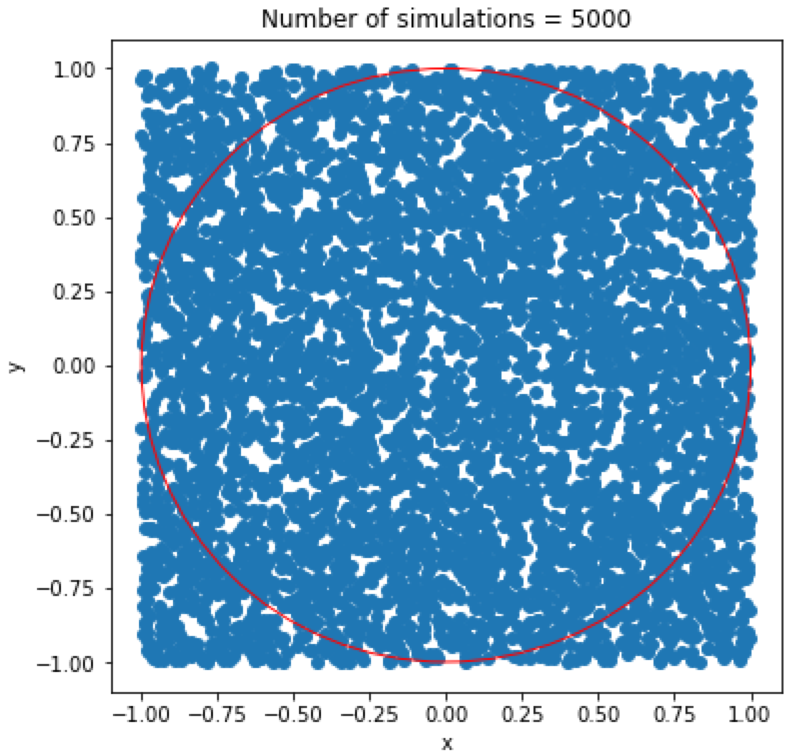
In [15]:

```
def simulate_pi(nsim):
    X = uniform.rvs(loc = -1, scale = 2, size = (nsim, 2)) #  the uniform distri

    fig = plt.figure(figsize = (6,6))
    ax = fig.add_subplot(111)
    plt.scatter(X[:, 0], X[:, 1])
    circle = plt.Circle((0, 0), 1, color='r', fill=False)
    ax.add_patch(circle)
    plt.title('Number of simulations = {}'.format(nsim))
```

```
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

    fraction = np.sum(X[:,0] ** 2 + X[:, 1] ** 2 <= 1) / nsim
    print("The fraction of samples falling in the circle is %.3f" % fraction)
    print("The estimated pi is %.3f" % (4*fraction))

np.random.seed(23)
simulate_pi(2000)

np.random.seed(123)
simulate_pi(5000)
```




```
The fraction of samples falling in the circle is 0.788
The estimated pi is 3.154
```




```
The fraction of samples falling in the circle is 0.788
The estimated pi is 3.152
```

Determine Value-at-Risk (VaR) and Expected Shortfall (ES)

Let r be the continuously compounded monthly return on a stock and W_0 be the initial wealth to be invested over the month. Assume that $r \sim N(0.01, 0.1^2)$ and $W_0 = \$10,000$.

Determine 5% VaR

Let W_1 be the end of month wealth and $L_1 \equiv W_1 - W_0$ be the profit of the investment, which can be negative. The monthly VaR on the W_0 investment with probability α is denoted by VaR_α such that

$$P(L_1 \leq VaR_\alpha) = \alpha.$$

Note that

$$L_1 = W_1 - W_0 = W_0 \cdot (e^r - 1).$$

Thus, we have

$$P\Big(W_0 \cdot (e^r - 1) \leq VaR_\alpha\Big) = \alpha.$$

It implies that

$$P(r \le x) = \alpha\ ,$$

where $x = \ln(\frac{VaR_\alpha}{W_0} + 1)$ or $VaR_\alpha = W_0 \cdot (e^x - 1)$. From the definition of quantile function (inverse CDF), we know $x = q^r_\alpha$. Thus, $VaR_\alpha = W_0 \cdot (e^{q^r_\alpha} - 1)$.

Therefore, to compute 5% VaR, we compute

1. $q^r_{5\%}$
2. $VaR_{5\%} = \$10,000 \cdot (e^{q^r_{5\%}} - 1)$

In [16]:

```
w0 = 10000
mu_r = 0.01
sigma_r = 0.1
alpha = 0.05

def VaR(alpha, w0 = w0, mu = mu_r, sigma = sigma_r):
    q_r = norm.ppf(alpha, mu, sigma)
    return w0 * (np.exp(q_r) - 1)

print("The 5% quantile of N({},{:.2f}) is {:.3f}".format(mu_r, sigma_r**2, norm.
print("The 5% value at risk is {:.3f}".format(VaR(alpha)) )
```

```
The 5% quantile of N(0.01,0.01) is -3.280
The 5% value at risk is -1431.440
```

Determine 5% ES

Given the profit is not larger than VaR_α, the expected shortfall (ES) at α measures the conditional expected profit, i.e.,

$$ES_\alpha = E[L_1 | L_1 \le VaR_\alpha] = \frac{\int_{-\infty}^{VaR_\alpha} x f_{L_1}(x) dx}{P(L_1 \le VaR_\alpha)} = \alpha^{-1} \int_0^\alpha VaR_u du.$$

Next, we introduce several ways to compute ES_α, where $\alpha = 5\%$.

In [17]:

```
import scipy.integrate as integrate

integrate.quad(VaR, 0, alpha)
```

Out[17]:

```
(-88.82667054220768, 8.17306187173017e-08)
```

In [18]:

```
ES = integrate.quad(VaR, 0, alpha)[0] / alpha
print("The 5% expected shortfall is {:.3f}".format (ES))
```

```
The 5% expected shortfall is -1776.533
```

Computations through MC simulations

The above conditional expectation can be approximated by its sample counterpart.

In [19]:

```
np.random.seed(123)
nsim = 5000
r = norm.rvs(loc = mu_r, scale = sigma_r, size = nsim)

X = w0 * (np.exp(r) - 1)
Y = (X <= VaR(alpha)) * 1

ES_sample = np.mean(Y * X)/np.mean(Y)
print("The simulated ES is %.3f" % ES_sample)
```

```
The simulated ES is -1759.723
```