In [1]:
```
import numpy as np
```

In [2]:
```
np.random.seed(1234)
np.random.uniform(0, 100), np.random.normal(size=(2,3))
# Mersenne Twister - pseudo random number generator
```

Out[2]:
```
(19.151945037889227, array([[-1.0336313 ,  2.02683258,  0.66250904],
        [ 0.67524634, -0.94029827, -0.95658428]]))
```

In [3]:
```
rng = np.random.default_rng(1234)
rng.uniform(0, 100), rng.normal(size=(2,3))
```

Out[3]:
```
(97.66997666981422, array([[ 0.06409991,  0.7408913 ,  0.15261919],
        [ 0.86374389,  2.91309922, -1.47882336]]))
```

In [4]:
```
# quasi random number generators
from scipy.stats import qmc
# sobol sequence
gen = qmc.Sobol(d=3, seed=1234)
gen.random_base2(m=2) # generate 2^m numbers
```

Out[4]:
```
array([[0.99361137, 0.28360828, 0.74058864],
       [0.03475311, 0.95930153, 0.21378241],
       [0.33277869, 0.02253785, 0.82184589],
       [0.6387977 , 0.7218523 , 0.34877372]])
```

In [5]:
```
# simulated option price
S0, K, T, r, q, vol = 180, 160, 0.5, 0.02, 0.015, 0.20
```

In [6]:
```
n = 100000
rng = np.random.default_rng(1234)
z = rng.standard_normal(size=n)
S = S0*np.exp((r-q-0.5*vol**2)*T +vol*np.sqrt(T)*z)
disc_payoffs = np.exp(-r*T)*np.maximum(S - K, 0.0)
sim_price = np.mean(disc_payoffs)
sim_price
```

Out[6]:
```
22.881914778296434
```

In [7]:
```
def simulated_price(S0, K, T, r, q, vol, n):
  z = rng.standard_normal(size=n)
  S = S0*np.exp((r-q-0.5*vol**2)*T +vol*np.sqrt(T)*z)
  disc_payoffs = np.exp(-r*T)*np.maximum(S - K, 0.0)
  sim_price = np.mean(disc_payoffs)
  std_error = np.std(disc_payoffs, ddof=1)/np.sqrt(n)
  return sim_price, std_error
```

In [8]:
```
simulated_price(S0, K, T, r, q, vol, n=1000000)
```

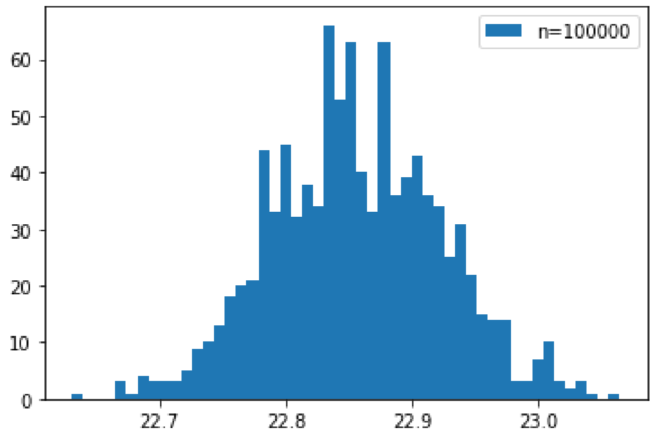
Out[8]:
```
(22.882269457296434, 0.021943525416597772)
```

In [9]:
```
# empirical distribution of simulated price
sim_prices = np.zeros(shape=1000)
for i in range(1000):
  sim_prices[i], _ = simulated_price(S0, K, T, r, q, vol, n)
```

In [10]:
```
import matplotlib.pyplot as plt
```

In [11]:
```
plt.hist(sim_prices, bins=50, label='n=100000')
plt.legend()
```

Out[11]: <matplotlib.legend.Legend at 0x7f9f5de2b0d0>



In [12]:
```
rng1 = np.random.default_rng(2468)
mean = [0.0, 0.0, 0.0] #[0.0]*3
corr12, corr13, corr23 = 0.9, 0.7, 0.4
cov = [[1.0, corr12, corr13],
       [corr12, 1.0, corr23],
       [corr13, corr23, 1.0]]

rng1.multivariate_normal(mean, cov, size=5)
```

Out[12]:
```
array([[ 0.63944433,  0.70663385,  0.66834461],
       [-0.87095412, -1.24886267,  0.30961717],
       [-0.48900968,  0.15799093, -1.35955048],
       [-0.29869728,  0.49668676, -1.18293483],
       [ 1.14910657,  0.78939262,  1.76635015]])
```

In [13]:
```
# Z = vector of independent random samples from standard normal distribution
z = rng1.standard_normal(size=5)
z
```

Out[13]:
```
array([-0.23206345, -0.43716652,  1.44722568, -0.07485793, -0.56905403])
```

In [13]:

Asian option

Assume that the underlying asset $S(t)$ follows a GBM

$$S(t_i) = S(t_{i-1})\exp((\mu - \sigma^2/2)\Delta t_i + \sigma\sqrt{\Delta t_i} Z_i)$$

In [14]:
```
# call option on daily average over month of March 2023
# option expiry = March 31st
import datetime as dt

today = dt.date(2023, 2, 17)
avg_start = dt.date(2023, 3, 1)
avg_end = dt.date(2023, 3, 31)
expiry = dt.date(2023, 3,31)
# S0 = price on Feb 17, 2023
S0, K, r, q, v = 180, 160, 0.02, 0.015, 0.20
n = 100000
```

In [15]:
```
#times = [(avg_start - today).days/365, (avg_start - today).days/365 + 1.0/365 ]
times = [((avg_start - today).days + i)/365 for i in range(31)]
T = times[-1]
T = (expiry - today).days/365
```

In [16]:
```
#times[:5], times[-1]
```

In [21]:
```
z = rng1.standard_normal(size=(31, n))
# asset price at t1
S = S0*np.exp((r - q - 0.5*v**2)*times[0] + v*np.sqrt(times[0])*z[0])
sum = S0*np.exp((r - q - 0.5*v**2)*times[0] + v*np.sqrt(times[0])*z[0])
for i in range(1, 31):
  S = S*np.exp((r - q - 0.5*v**2)*(times[i] - times[i-1]) + v*np.sqrt(times[i] - t
imes[i-1])*z[i])
  sum += S
A = sum / 31
disc_payoffs = np.exp(-r*T)*np.maximum(A - K,0)
price = np.mean(disc_payoffs)
stdev = np.std(disc_payoffs, ddof=1)
price, stdev
```

Out[21]:
```
(20.032722051289667, 8.720821340190453)
```

In [ ]:
```
# if antithetical sampling is used,
# the variance of the MC estimator will be reduced
```