Financial Derivatives
Lecture 11: Numerical methods in pricing derivatives

Douglas Chung

National Chengchi University

1 Python examples

1.1 Estimate the value of π

$\forall x, y \in \mathbb{R}^2 : [-1, 1]$

Area of square:

$$2 \times 2 = 4$$

Circle with a center at $\{0, 0\}$:

$$x^2 + y^2 \leq 1$$

its area is given by:

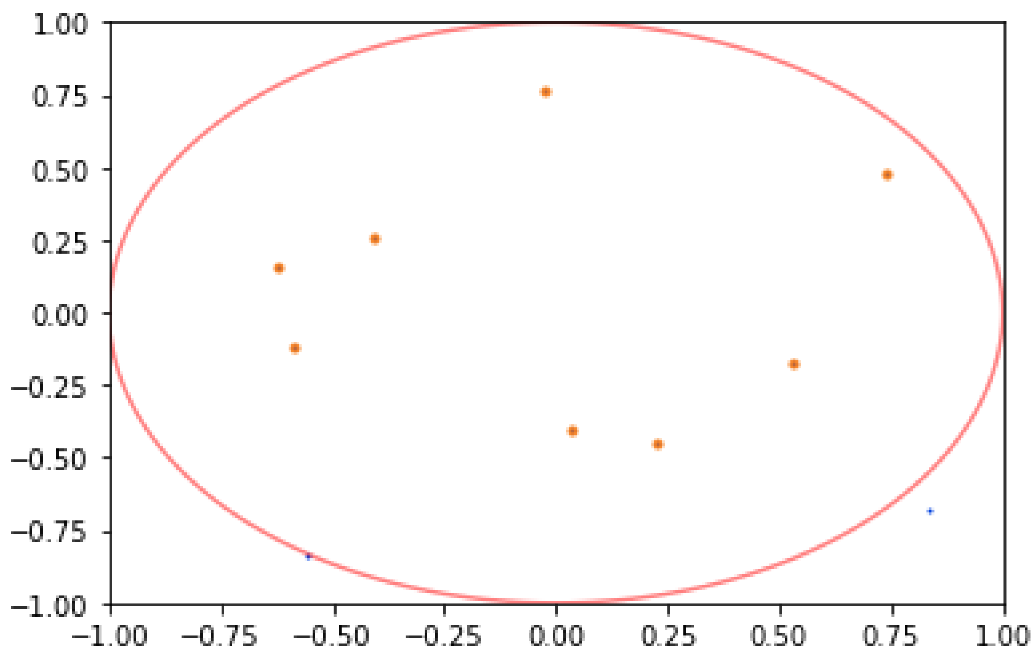
$$\pi r^2 = \pi$$

Pi is estimated by:

$$\hat{\pi} = \frac{4 \times \text{counts in circle}}{\text{counts in triangle}}$$

[1]:
```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(5)

n = 10

x = np.random.uniform(-1,1,n)
y = np.random.uniform(-1,1,n)
z = x**2 + y**2

fig, ax = plt.subplots()
```

```
dim = 1

plt.xlim(-dim,dim)
plt.ylim(-dim,dim)
plt.scatter(x,y,s=1,c='xkcd:blue')

plt.scatter(x[z<1],y[z<1],s=10,c='xkcd:orange',alpha=0.75)
a = np.linspace(-1.0, 1.0, 100)
b = np.linspace(-1.0, 1.0, 100)
X, Y = np.meshgrid(a,b)
F = X**2 + Y**2 - 1
plt.contour(X,Y,F,[0],colors='r',alpha=0.5)
plt.show()

print('Pi is:')
4*sum(z<1)/n
```



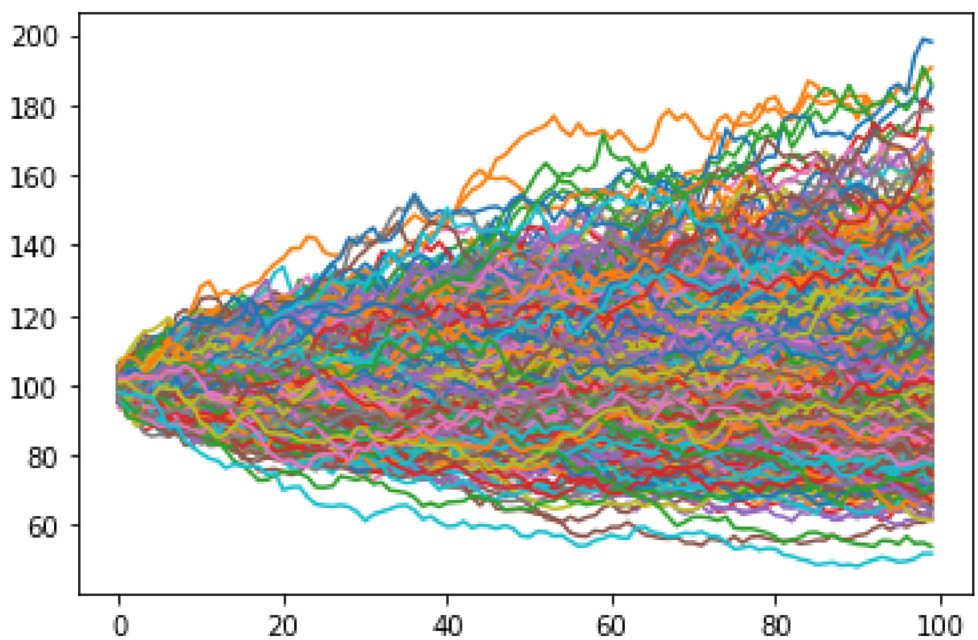

```
Pi is:
```

```
[1]: 3.2
```

1.2 Pricing options with Monte Carlo simulation

Step 1: simulate logarithm stock price

[2]:
```
import numpy as np

np.random.seed(5)

nsim = 1000

nstep = 100
T = 1
dt = T/nstep

r = 0.05
sigma = 0.2

S_0 = 100

e = np.random.normal(loc=0.0, scale=1.0, size=(nsim,nstep))

S_t = S_0*np.exp((r - sigma ** 2 / 2) * dt + sigma * np.sqrt(dt) * e).
↪cumprod(axis=1)
plt.plot(S_t.T)
plt.show()
```



Step 2: compute terminal payoffs and prices for European options

[3]:
```
K = 100

S_T = S_t[:,nstep-1]

C_T = np.fmax(S_T - K,0)
C_0 = np.exp(-r*T) * np.mean(C_T)
print('European call price (Monte Carlo)')
print(round(C_0,2))

P_T = np.fmax(K - S_T,0)
P_0 = np.exp(-r*T) * np.mean(P_T)
print('European put price (Monte Carlo)')
print(round(P_0,2))
```

```
European call price (Monte Carlo)
11.09
European put price (Monte Carlo)
5.41
```

Step 3: compare with the Black-Scholes-Merton model

[4]:
```
from scipy import stats

def BS_call(S, K, r, t, Sigma):
    d1 = (np.log(S/K) + (r + 0.5 * Sigma**2)*t)/(Sigma * np.sqrt(t))
    d2 = d1 - Sigma * np.sqrt(t)
    Call = S * stats.norm.cdf(d1,0.0,1.0) - K * np.exp(-r*t) * stats.norm.
 ↪cdf(d2,0.0,1.0)
    return Call

def BS_call_delta(S, K, r, t, Sigma):
    d1 = (np.log(S/K) + (r + 0.5 * Sigma**2)*t)/(Sigma * np.sqrt(t))
    delta = stats.norm.cdf(d1,0.0,1.0)
    return delta

def BS_put(S, K, r, t, Sigma):
    d1 = (np.log(S/K) + (r + 0.5 * Sigma**2)*t)/(Sigma * np.sqrt(t))
    d2 = d1 - Sigma * np.sqrt(t)
    Put = K * np.exp(-r*t) * stats.norm.cdf(-d2,0.0,1.0) - S * stats.norm.
 ↪cdf(-d1,0.0,1.0)
    return Put

print('European call price (BSM)')
print(round(BS_call(S_0, K, r, T, sigma),2))
print('European put price (BSM)')
print(round(BS_put(S_0, K, r, T, sigma),2))
```

```
European call price (BSM)
```

```
10.45
European put price (BSM)
5.57
```

Step 4: estimate Δ using Monte Carlo simulation

[5]:
```
h = 0.0001
Sh_0 = S_0 + h
Sh_t = Sh_0*np.exp((r - sigma ** 2 / 2) * dt + sigma * np.sqrt(dt) * e).
 ↪cumprod(axis=1)

Sh_T = Sh_t[:,nstep-1]

Ch_T = np.fmax(Sh_T - K,0)
Ch_0 = np.exp(-r*T) * np.mean(Ch_T)

delta_mc = (Ch_0 - C_0)/h
print('European call delta (MC)')
print(round(delta_mc, 2))

print('European call delta (BSM)')
print(round(BS_call_delta(S_0, K, r, T, sigma), 2))
```

```
European call delta (MC)
0.65
European call delta (BSM)
0.64
```

1.3 Monte Carlo Puzzle I

[6]:
```
prob = 10e-12
ntrial = 1000000

X = np.random.uniform(0,1,ntrial)

np.mean((X < prob)*1)
```

[6]:
```
0.0
```

1.4 Monte Carlo Puzzle II

[7]:
```
nsim = 100

nstep = 1
T = 5
dt = T/nstep

r = 0.05
sigma = 0.1

S_0 = 100

e = np.random.normal(loc=0.0, scale=1.0, size=(nsim,nstep))

S_t = S_0*np.exp((r - sigma ** 2 / 2) * dt + sigma * np.sqrt(dt) * e).
↪cumprod(axis=1)
S_T = S_t[:,nstep-1]
ES_t = np.exp(-r*T) * np.mean(S_T)

print('The present value of expected stock price:')
print(round(ES_t,2))
```

```
The present value of expected stock price:
100.48
```

1.5 Antithetic variates

[11]:
```
import numpy as np
import pandas as pd

np.random.seed(5)

nsim = 1000

nstep = 100
T = 1
dt = T/nstep

r = 0.05
sigma = 0.2

S_0 = 100

e = np.random.normal(loc=0.0, scale=1.0, size=(nsim,nstep))
print('The empirical average of shocks:')
print(round(np.mean(e),4))
```

```
e_av = np.concatenate((e,-e), axis = 0)
print('The empirical average of shocks after the method of antithetic variates:')
print(round(np.mean(e_av),4))

K = 100

S_t = S_0*np.exp((r - sigma ** 2 / 2) * dt + sigma * np.sqrt(dt) * e_av).
↪cumprod(axis=1)
S_T = S_t[:,nstep-1]

C_T = np.fmax(S_T - K,0)
C_0 = np.exp(-r*T) * np.mean(C_T)
print('European call price (Monte Carlo with antithetic variates)')
print(round(C_0,2))

P_T = np.fmax(K - S_T,0)
P_0 = np.exp(-r*T) * np.mean(P_T)
print('European put price (Monte Carlo with antithetic variates)')
print(round(P_0,2))

print('European call price (BSM)')
print(round(BS_call(S_0, K, r, T, sigma),2))

print('European put price (BSM)')
print(round(BS_put(S_0, K, r, T, sigma),2))
```

```
The empirical average of shocks:
0.0038
The empirical average of shocks after the method of antithetic variates:
0.0
European call price (Monte Carlo with antithetic variates)
10.6
European put price (Monte Carlo with antithetic variates)
5.68
European call price (BSM)
10.45
European put price (BSM)
5.57
```