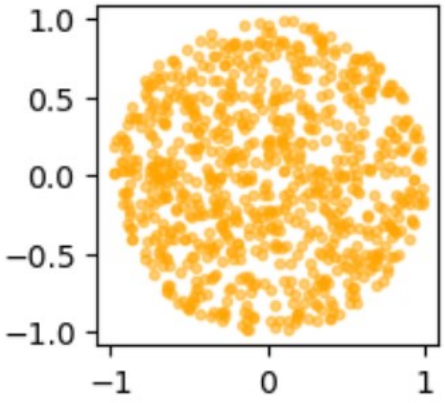
Q1.

Estimating the value of Pi using Monte Carlo Simulations

We can estimate the value of Pi using Monte Carlo technique. The procedure is based on probabilities and random number generation. We consider a square extending from x=-1 to x=1 and y=-1 to y=1 such that each side is of length 2 and a circle of radius 1 is inscribed in it such that center of both the square and the circle lie at the origin.



Let N be the total number of randomly generated points. If the points generated happen to lie inside the circle then we store the coordinates of the points in the array named "inside". Since the generation of points is completely random and the chances of a point lying at any position is equally likely,

Therefore, the probability of a point lying inside the circle will be directly proportional to the area of circle . Thus

$$\frac{N_{inside}}{N} = \frac{Area\ of\ circle}{Area\ of\ square} = \frac{\pi r^2}{a^2} = \frac{\pi}{4}$$

$$\pi = 4\,\frac{N_{inside}}{N}$$

N_{inside} = Points lying inside the circle

N = Total number of points generated

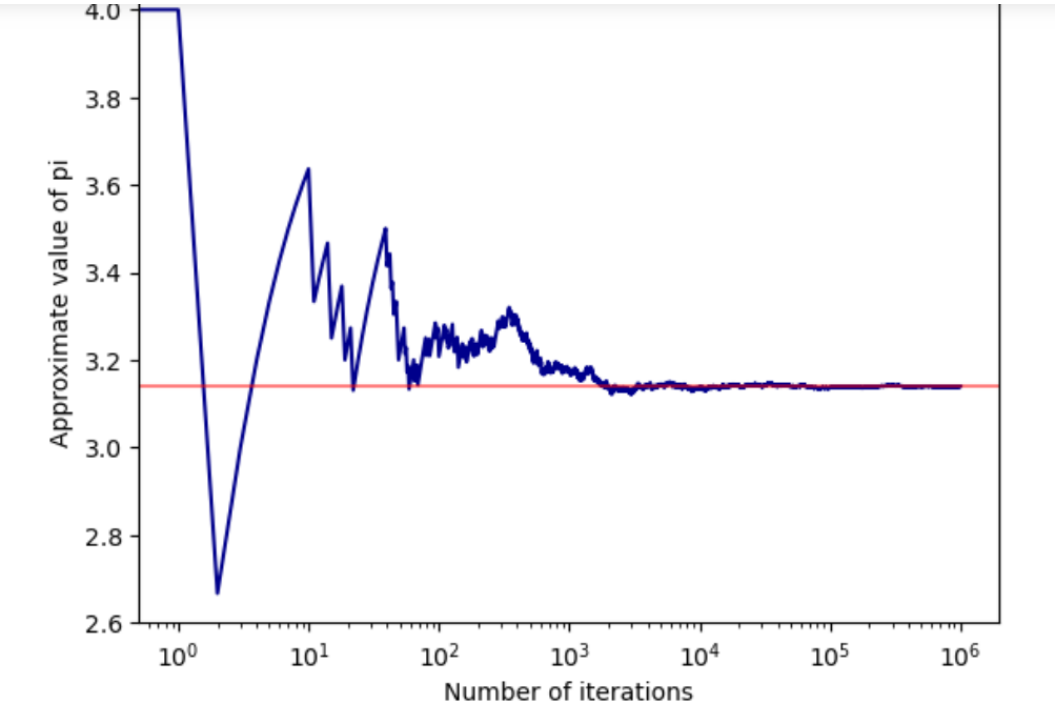
Code Snippet

```
N = 1000000
inside = []
for i in range(N):
    x = numpy.random.uniform(-1, 1)
    y = numpy.random.uniform(-1, 1)
    if numpy.sqrt(x**2 + y**2) < 1:
        inside.append((x, y))

plt.figure(figsize=(2, 2))
plt.scatter([x[0] for x in inside], [x[1] for x in inside], marker=".", alpha=0.5,color="orange")

print("Value of pi = ",4 * len(inside)/float(N))
```

```
Value of pi =  3.140432
```

Here, N denotes the total number of points generated that lie inside square. The approximation Converges to the mathematical value of pi if we have very large number of points ideally tending to infinite points inside the square.

Convergence of approximations to Mathematical Value



Q2.

a) Given : Current stock price (at t=0) = 100 rupees

Expected value of stock price after:

i) 1 minute = 100
ii) 10 minutes = 100
iii) 1 hour = 100
iv) 1 month =100

Every minute stock price moves up and down with same probability p=q= $\frac{1}{2}$

Expected move in 1 minute = $\left(\frac{1}{2}\right)(1) + \left(\frac{1}{2}\right)(-1) = 0$

{ Here , 1 and -1 are for upward and downward movement of stock }

In any minute the expected movement of stock = 0

Therefore the stock price will remain equal to 100 rupees at all instants.

b) (i) The probability of the stock price going to 102 before going to 96, will be 2/3 = 0.6667

Let X = p(102,96) is the required probability , where p(a,b) is the probability of reaching a before b.
In one timestep the stock price either moves upward or downward so there is an equal probability of being at 1 or -1.
If stock price moves up ,the probability of reaching 102 first will now be p(101,95), and if stock moves down the probability will be p(103,97)
So X= [p(101,95) + p(103,97)] $\frac{1}{2}$

In next timestep there is 25% chance to be at -2 , 25% chance to be at 2 and 50% chance to be back at 0.

X = [p(100,94) + 2p(102,96) + p(104,98)] $\frac{1}{4}$

4X = p(100,96) +2X + p(104,98)

2X = 1 + p(104,98) **let z = p(104,98)**

2X = 1 + z ……(i)

Following same procedure for z for 2 more steps

Z = (p(103,97)+p(105,99))$\frac{1}{2}$

Z = (p(106,100)+2p(104,98)+p(102,96))$\frac{1}{4}$

4Z = 0 + 2Z + X

Z = $\frac{x}{2}$ ……..(ii)

Substituting equation (ii) in equation (i)

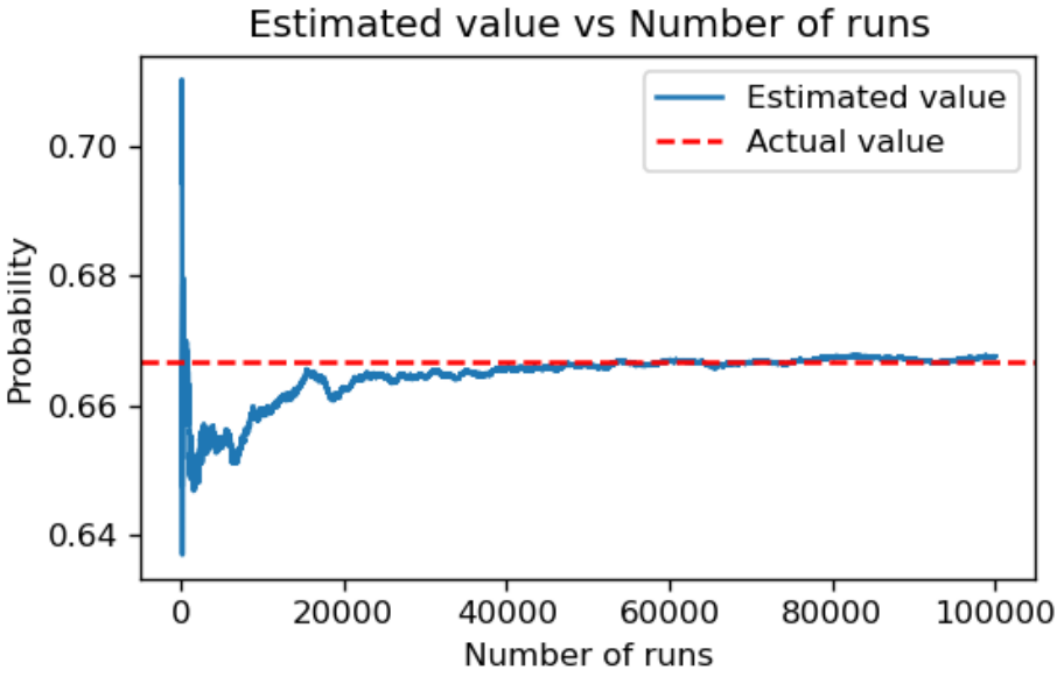
2X = 1 + X/2

X = 2/3

Thus, the probability of the stock price going to 102 before going to 96 is $\frac{2}{3}$ = 0.6667

(ii) Code is provided in **JPMC_CaseStudy_Group16.ipynb**

```
Number of runs = 6515
Probability is 0.66748
```



Q3. Black Scholes Model for Option Pricing

Black Scholes formula for call option

$$C = N(d_1)S_t - N(d_2)Ke^{-rt}$$

$$\text{where } d_1 = \frac{\ln \frac{S_t}{K} + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}$$

$$\text{and } d_2 = d_1 - \sigma\sqrt{t}$$

Given :

Strike: 180
Time To expiration: 30 days (1/12th of a year)
Implied Vol: 15%
Interest Rate: 2%
Current Stock Price: 200

Code snippet

```
N = norm.cdf

S = 200             #current stock price
K = 180             #strike
T = 1/12            #time to expiration
r = 0.02            #risk free interest rate
sigma = 0.15        #implied volatility

# Call option price using Black scholes formula
def BS_CALL():
    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S * N(d1) - K * np.exp(-r*T)* N(d2)

# Put option price using Black scholes formula
def BS_PUT():
    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma* np.sqrt(T)
    return K*np.exp(-r*T)*N(-d2) - S*N(-d1)

print("Call option Price : ",BS_CALL())
print("Put option Price : ",BS_PUT())
```

Output :

```
Call option Price :  20.317700916589644
Put option Price :  0.017950777758622527
```

Code is provided in **JPMC_CaseStudy_Group16.ipynb**

Q4. Option Pricing using Monte Carlo Simulations

The Black Scholes model is based on Geometric Brownian Motion (GBM) with constant drift and volatility. The underlying stock S follows a GBM which satisfies the SDE

$dS_t = \mu S_t dt + \sigma S_t dW_t$

By using Ito's lemma, the analytical solution of the SDE is obtained as

$$S(\Delta t) = S(0)\exp\left[\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \left(\sigma\sqrt{\Delta t}\right)\varepsilon\right]$$

Or **S(t) = S(t-1)*exp((r-0.5*sigma**2)*dt + sigma*N(0,1)*sqrt(dt))**

Where ε is a standard normal variable.

Geometric Brownian motion assumes infinitely divisible time throughout the life of the option so we sample 240 increments over a period of 30 days ie dt = T/N

Thus dt = $\frac{1}{12\times240}$

Here we use Monte carlo methods to analyse the value of an option.

Firstly, we simulate the price of the underlying asset by random number generation for a number of paths .

To create the different paths, we begin by utilizing the function np.random.standard_normal that draw (N+1)×I samples from a standard Normal distribution.

For a simulation with ‘I’ paths even though all start with a value of 200, the stock's final value varies with a volatility of 15%.

```
def gen_paths(S0, r, sigma, T, N, I):
    dt = float(T) / N
    paths = np.zeros((N + 1, I), np.float64)
    paths[0] = S0
    for t in range(1, N + 1):
        rand = np.random.standard_normal(I)
        paths[t] = paths[t - 1] * np.exp((r - 0.5 * sigma ** 2) * dt +
                                         sigma * np.sqrt(dt) * rand)
    return paths
```

Each path consists of a list of stock prices. The final stock value obtained in each path is the stock price at expiration (S_t)

```
# generate paths
np.random.seed(123)
paths = gen_paths(S0, r, sigma, T, N, i)

# paths[-1] gives the value of stock at expiration (St)
V = np.maximum(0, paths[-1] - K)
```

We first calculate the payoff for call option using **max(0,S_t-K)** for all paths and then calculate the average of the payoffs obtained .

The final estimated option price at t=0 is the discounted value of the average of payoffs obtained in the above step.

Since for European options the option is exercised at maturity . Thus, the call option price at t=0 is given by

$$C_0 = e^{-rT} \frac{1}{N} \sum_{i}^{N} \max(S_T - K, 0)$$

```
# The payoff will be the max of (St - K, 0)
CallPayoffAverage = np.average(np.maximum(0, paths[-1] - K))
CallPayoff = discount_factor * CallPayoffAverage

print("Call option price :",CallPayoff)
```

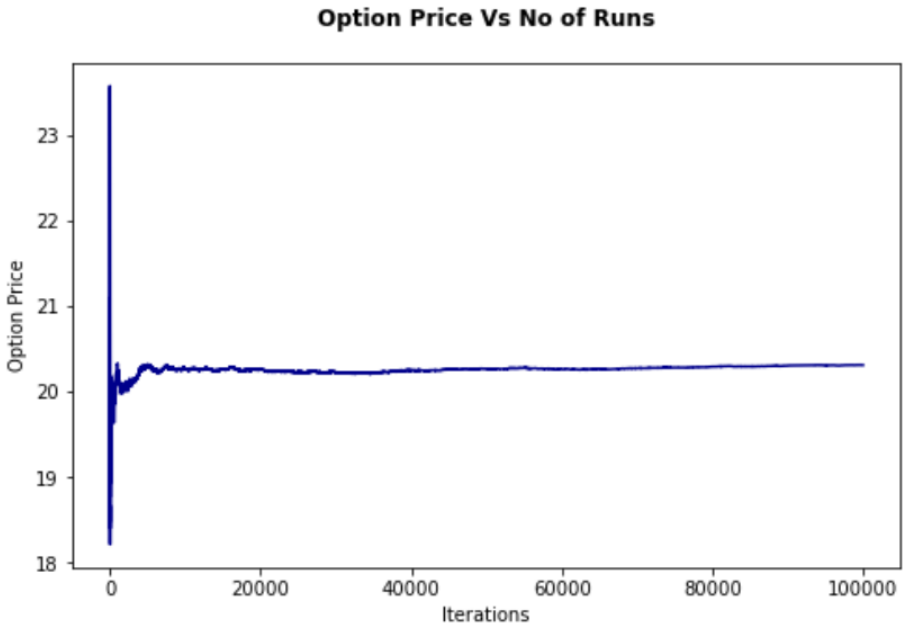
```
Call option price : 20.31684431668016
```

Call Option Price obtained from:

Black Scholes Model **: 20.3177009**

Monte Carlo simulations (with I = 1e6) **: 20.3168443**

As we increase N towards infinity the price approaches the Black-Scholes price due to Central Limit Theorem.



Code is provided in **JPMC_CaseStudy_Group16.ipynb**