In [1]:
```python
import numpy as np
import pandas as pd
```

Simple code to Price an European call and put option using Monte carlo simulation

Pricing a Asian option(Path dependent option).

In [2]:
```python
def GBM(S0, rf, T, vol, nsteps, nsims, K):
    dt = T/nsteps
    S = np.zeros((nsims, nsteps+1))
    S[:, 0] = S0

    for i in range(nsteps):
        phi = np.random.randn(nsims)
        S[:, i+1] = S[:, i] * np.exp((rf - 0.5 * vol * vol) * dt + vol * phi * np.sqrt(dt))

    df_S = pd.DataFrame(S)
    S_avg = df_S.iloc[:, 1:].mean(axis=1)

    payoff_call_european = np.maximum(df_S.iloc[:, -1] - K, 0)
    price_call_european = np.exp(-rf * T) * payoff_call_european.mean()

    payoff_put_european = np.maximum(K - df_S.iloc[:, -1] , 0)
    price_put_european = np.exp(-rf * T) * payoff_put_european.mean()

    payoff_asian = np.maximum(S_avg - K, 0)
    price_asian = (np.exp(-rf * T) * payoff_asian).mean()

    return price_call_european, price_put_european, price_asian
```

In [3]:
```python
## Input Parameters
S0 = 200
rf = 0.05
T = 1
vol = 0.2
nsteps = 252
nsims = 20000
K = 185

european_call_price,european_put_price, asian_price, = GBM(S0, rf, T, vol, nsteps, nsims, K)
print("European Call Option Price:", european_call_price)
print("European Put Option Price:", european_put_price)
print("Asian Option Price:", asian_price)
```

```
European Call Option Price: 29.848200482476035
European Put Option Price: 5.917014390922322
Asian Option Price: 21.3636471595102
```

Finite difference approximation to calculating greeks - Delta, Gamma and Vega

In [4]:
```python
def GBM(S0, rf, T, vol, nsteps, nsims, K, random_seed=42):
    np.random.seed(random_seed)
    dt = T / nsteps
    S = np.zeros((nsims, nsteps + 1))
    S[:, 0] = S0

    for i in range(nsteps):
        phi = np.random.randn(nsims)
        S[:, i + 1] = S[:, i] * np.exp((rf - 0.5 * vol * vol) * dt + vol * phi * np.sqrt(dt))

    df_S = pd.DataFrame(S)

    payoff_call_european = np.maximum(df_S.iloc[:, -1] - K, 0)
    european_call_price = np.exp(-rf * T) * payoff_call_european.mean()

    return european_call_price, df_S,phi


def Finite_Difference_Delta(epsilon=1, method='centered'):
    # Unpack the tuple from GBM
    price, simulated_data,phi = GBM(S0, rf, T, vol, nsteps, nsims, K)

    price_up = GBM(S0 + epsilon, rf, T, vol, nsteps, nsims, K)[0]
    price_down = GBM(S0 - epsilon, rf, T, vol, nsteps, nsims, K)[0]

    if method == 'forward':
        delta = (price_up - price) / epsilon
        return delta
    elif method == 'backward':
        delta = (price - price_down) / epsilon
        return delta
    else:
        delta = (price_up - price_down) / (2 * epsilon)
        return delta
```

```python
def Finite_Difference_Gamma(epsilon=1):
    # Unpack the tuple from GBM
    price, simulated_data,phi = GBM(S0, rf, T, vol, nsteps, nsims, K)

    price_up = GBM(S0 + epsilon, rf, T, vol, nsteps, nsims, K)[0]
    price_down = GBM(S0 - epsilon, rf, T, vol, nsteps, nsims, K)[0]

    gamma = (price_up - 2 * price + price_down) / epsilon ** 2
    return gamma


def Finite_Difference_Vega(epsilon=0.0001, method='centered'):
    # Unpack the tuple from GBM
    price, simulated_data,phi = GBM(S0, rf, T, vol, nsteps, nsims, K)

    price_up = GBM(S0, rf, T, vol + epsilon, nsteps, nsims, K)[0]
    price_down = GBM(S0, rf, T, vol - epsilon, nsteps, nsims, K)[0]

    if method == 'forward':
        vega = price_up - price
        return vega
    elif method == 'backward':
        vega = price - price_down
        return vega
    else:
        vega = (price_up - price_down) / (2 * epsilon)
        return vega


# # Example usage
# S0 = 200
# rf = 0.05
# T = 252/365
# vol = 0.25
# nsteps = 252
# nsims = 50000
# K = 180

price, simulated_data,random_numbers = GBM(S0, rf, T, vol, nsteps, nsims, K, random_seed=42)
d = Finite_Difference_Delta()
g = Finite_Difference_Gamma()
v = Finite_Difference_Vega()
print("European Call Option Price:", price)
print("Finite Difference Delta:", d)
print("Finite Difference Gamma:", g)
print("Finite Difference Vega:", v)
```

```
European Call Option Price: 29.858048952868973
Finite Difference Delta: 0.7692234196647334
Finite Difference Gamma: 0.006960238060550239
Finite Difference Vega: 60.494783472346825
```

Calculating Option Greeks using Likelihood ratio method

In [6]:

```python
"""The likelihood ratio method
provides an alternative approach to derivative estimation requiring no smoothness
at all in the discounted payoff and thus complementing the pathwise
method. It accomplishes this by differentiating probabilities rather than payoffs. As with the pathwise method,
the validity of this approach relies on an
interchange of differentation and integration. Probability densities are typically smooth functions of their pa
option payoffs are not"""


def Likelihood_Ratio_Method(output = 'price'):
    a = np.log(simulated_data.iloc[:, -1]/ S0)
    b = (rf - 0.5* (vol**2))*T
    zeta = ((a - b) / vol*np.sqrt(T))
    indicator_fn = np.where(simulated_data.iloc[:, -1] > K, 1, 0)
    discounting_factor = np.exp(-rf * T)
    terminal_price = simulated_data.iloc[:, -1]
    discounted_payoff = indicator_fn * discounting_factor * (terminal_price - K)

    if output == 'vega':
        c = (rf + 0.5* (vol**2))*T
        del_zeta = (np.log(S0 / simulated_data.iloc[:, -1]) + (c)) / (vol**2)*np.sqrt(T)
        score_fn = -((1/vol) + (zeta * del_zeta))
        vega = (discounted_payoff * score_fn).mean()
        return vega

    elif output == 'delta':
        c = S0* (vol**2)*T
        d = (a - b)/c
        delta = (discounted_payoff * d ).mean()
        return delta

    elif output == 'gamma':
```

```
        m = np.log((simulated_data.iloc[:, -1])/ S0)
        n = (rf - 0.5* (vol**2))*T
        zeta = ((m - n)/ vol*np.sqrt(T))
        p = 1 / ((S0)**2 * vol*np.sqrt(T))
        gamma = ((p * (((zeta**2 - 1)/vol*np.sqrt(T)) - zeta)) * discounted_payoff).mean()
        return gamma


    else:
        price = discounted_payoff.mean()
        return price



Price = Likelihood_Ratio_Method()
Delta = Likelihood_Ratio_Method('delta')
Gamma = Likelihood_Ratio_Method('gamma')
Vega = Likelihood_Ratio_Method('vega')


print("European Call Option Price:", Price)
print("Likelihood Ratio Delta:", Delta)
print("Likelihood Ratio Gamma:", Gamma)
print("Likelihood Ratio Vega:", Vega)
```

```
European Call Option Price: 29.858048952868966
Likelihood Ratio Delta: 0.7745442432173576
Likelihood Ratio Gamma: 0.008022978020534952
Likelihood Ratio Vega: 64.18382416427977
```

Calculating Greeks using Pathwise Sensitivity method

In [7]:
```
# S0 = 100
# rf = 0.05
# T = 1
# vol = 0.2
# nsteps = 252
# nsims = 20000
# K = 100
"""discontinuities in a payoff are the main obstacle
to the applicability of the pathwise method. A simple rule of thumb states that
the pathwise method applies when the payoff is continuous in the parameter
of interest. In some cases, this obstacle can be
overcome by smoothing discontinuities through conditional expectations
"""


def Pathwise_Sensitivity_Method(output = 'price'):

    indicator_fn = np.where(simulated_data.iloc[:, -1] > K,1,0)
    discounting_factor = np.exp(-rf * T)
    Terminal_price = simulated_data.iloc[:, -1]
    discounted_payoff = indicator_fn * discounting_factor * (Terminal_price - K)
    discounted_indicator_fn = (indicator_fn) * np.exp(-rf * T)

    if output == 'vega':
        a = np.log((Terminal_price)/ S0)
        b = (rf + 0.5* (vol**2))*T
        #Z = random_numbers
        vega = (Terminal_price * discounted_indicator_fn * ((a - b)/vol)).mean()
        return vega

    elif output == 'delta':
        a = (Terminal_price)/ S0
        delta = (a * discounted_indicator_fn).mean()
        return delta

    else:
        price = discounted_payoff.mean()
        return price



Price = Pathwise_Sensitivity_Method()
Delta = Pathwise_Sensitivity_Method('delta')
Vega = Pathwise_Sensitivity_Method('vega')
print("European Call Option Price:", Price)
print("Pathwise Sensitivity Delta:", Delta)
print("Pathwise Sensitivity Vega:", Vega)
```

```
European Call Option Price: 29.858048952868966
Pathwise Sensitivity Delta: 0.769522744495107
Pathwise Sensitivity Vega: 60.49200262270165
```

In [ ]:
```
# Mixed approach - LR-PW for Gamma calculation

# indicator_fn = np.where(simulated_data.iloc[:, -1] > K,1,0)
# a = np.log(simulated_data.iloc[:, -1]/ S0)
```

```
# b = (rf - 0.5* (vol**2))*T
# zeta = ((a - b) / vol*np.sqrt(T))
# discounting_factor = np.exp(-rf * T)
# gamma = (discounting_factor * (zeta/(S0**2*vol*np.sqrt(T))) * indicator_fn * K).mean()
# gamma
```

Combined code for Option greeks calculation using Likelihood ratio and Pathwise sensitivity method

In [8]:
```
def Display_Greeks(method= None):
    if method == 'Likelihood_Ratio':
        def Likelihood_Ratio_Method(output='price'):
            a = np.log((simulated_data.iloc[:, -1]) / S0)
            b = (rf - 0.5 * (vol ** 2)) * T
            discounted_payoff = (np.maximum(simulated_data.iloc[:, -1] - K, 0)) * (np.exp(-rf * T))

            if output == 'vega':
                c = (rf + 0.5 * (vol ** 2)) * T
                d = -(1 / vol)
                e = d + ((a - b) / vol * np.sqrt(T)) * ((a - c) / vol ** 2 * np.sqrt(T))
                vega = (discounted_payoff * e).mean()
                return vega

            elif output == 'delta':
                c = S0 * (vol ** 2) * T
                d = (a - b) / c
                delta = (discounted_payoff * d).mean()
                return delta

            elif output == 'gamma':
                m = np.log((simulated_data.iloc[:, -1])/ S0)
                n = (rf - 0.5* (vol**2))*T
                zeta = ((m - n)/ vol*np.sqrt(T))
                p = 1 / ((S0)**2 * vol*np.sqrt(T))
                gamma = ((p * (((zeta**2 - 1)/vol*np.sqrt(T)) - zeta)) * discounted_payoff).mean()
                return gamma

            else:
                price = discounted_payoff.mean()
                return price

        price = Likelihood_Ratio_Method()
        Delta = Likelihood_Ratio_Method('delta')
        Vega = Likelihood_Ratio_Method('vega')
        Gamma = Likelihood_Ratio_Method('gamma')

        print("European Call Option Price:", price)
        print("Likelihood Ratio Delta:", Delta)
        print("Likelihood Ratio Gamma:", Gamma)
        print("Likelihood Ratio Vega:", Vega)

    elif method == 'Pathwise_Sensitivity':
        def pathwise_sensitivity_method(output='price'):
            # Your logic for calculating option metrics
            indicator_fn = np.where(simulated_data.iloc[:, -1] > K, 1, 0)
            discounting_factor = np.exp(-rf * T)
            terminal_price = simulated_data.iloc[:, -1]
            discounted_payoff = indicator_fn * discounting_factor * (terminal_price - K)
            discounted_indicator_fn = indicator_fn * np.exp(-rf * T)

            if output == 'vega':
                a = np.log((terminal_price) / S0)
                b = (rf + 0.5 * (vol ** 2)) * T
                vega = (terminal_price * discounted_indicator_fn * ((a - b) / vol)).mean()
                return vega

            elif output == 'delta':
                a = (terminal_price) / S0
                delta = (a * discounted_indicator_fn).mean()
                return delta

            else:
                price = discounted_payoff.mean()
                return price

        # Call the function to get the desired output
        price = pathwise_sensitivity_method()
        delta = pathwise_sensitivity_method('delta')
        vega = pathwise_sensitivity_method('vega')

        # Print the results
        print("European Call Option Price:", price)
        print("Pathwise Sensitivity Delta:", delta)
        print("Pathwise Sensitivity Vega:", vega)
```

In [9]:
```
Display_Greeks(method='Pathwise_Sensitivity')
Display_Greeks(method='Likelihood_Ratio')
```

```
European Call Option Price: 29.858048952868966
Pathwise Sensitivity Delta: 0.769522744495107
Pathwise Sensitivity Vega: 60.49200262270165
European Call Option Price: 29.858048952868966
Likelihood Ratio Delta: 0.7745442432173576
Likelihood Ratio Gamma: 0.008022978020534952
Likelihood Ratio Vega: 64.18382416427973
```

In [ ]: