Submission Number: 2

Group Number: 14

Group Members:

Full Legal Name	Location (Country)	E-Mail Address	Non-Contributing Member (X)
Narsihma Reddy Dharmaiahgari Paripati	India	narsimha2506@gmail.com	
Sylendra Ruthwik Naidu	India	pm18naidus@iimidr.ac.in	
Ishaan Narula	India	ishaan.narula@outlook.com	

Statement of integrity: By typing the names of all group members in the text box below, you confirm that the assignment submitted is original work produced by the group (*excluding any non-contributing members identified with an "X" above*).

Narsihma Reddy Dharmaiahgari Paripati, Sylendra Ruthwik Naidu, Ishaan Narula

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed.

N/A

** Note, you may be required to provide proof of your outreach to non-contributing members upon request.*

MScFE 630 Computational Finance

Group #: 14 Submission #: 2

Note: The code for questions 1-4 are implemented in the attached notebook file named *"MScFE 630 - Group 14 - Submission 2 - Qs 1-4.ipynb"* and for questions 5-7 can be found in the file *"MScFE 630 - Group 14 - Submission 2 - Qs 5-7 v3 F.ipynb"*. Please ensure the latter is in the same folder as the Excel file *"Call Option Data.xlsx"*.

Answer 1

The Analytical calculation answer is 13.7348

In [18]:
```
#share specific information
S0 = 100
v0 = 0.06
kappa = 9
theta = 0.06
r = 0.08
sigma = 0.3
rho = -0.4

#Call option specific information
K = 100
T = 1
k_log = np.log(K)

#Approximation information
t_max = 30
N = 100
```

In [19]:
```
# Characteristic function code
a = sigma**2/2

def b(u):
    return kappa - rho*sigma*1j*u

def c(u):
    return -(u**2+1j*u)/2

def d(u):
    return np.sqrt(b(u)**2-4*a*c(u))

def xminus(u):
    return (b(u)-d(u))/(2*a)

def xplus(u):
    return (b(u)+d(u))/(2*a)

def g(u):
    return xminus(u)/xplus(u)

def C(u):
    val1 = T*xminus(u)-np.log((1-g(u)*np.exp(-T*d(u)))/(1-g(u)))/a
    return r*T*1j*u + theta*kappa*val1
```

Figure 1.1 Code piece of calculation of vanilla call option - part of initialization and function definitions

```
def D(u):
    val1 = 1-np.exp(-T*d(u))
    val2 = 1-g(u)*np.exp(-T*d(u))
    return (val1/val2)*xminus(u)

def log_char(u):
    return np.exp(C(u) + D(u)*v0 + 1j*u*np.log(S0))

def adj_char(u):
    return log_char(u-1j)/log_char(-1j)
```

```
In [20]: delta_t = t_max/N
from_1_to_N = np.linspace(1,N,N)
t_n = (from_1_to_N-1/2)*delta_t
```

```
In [21]: first_integral = sum((((np.exp(-1j*t_n*k_log)*adj_char(t_n)).imag)/t_n)*delta_t)
second_integral = sum((((np.exp(-1j*t_n*k_log)*log_char(t_n)).imag)/t_n)*delta_t)
```

```
In [22]: fourier_call_val = S0*(1/2 + first_integral/np.pi)-np.exp(-r*T)*K*(1/2 + second_integral/np.pi)
print(fourier_call_val)
```

```
13.734895692109049
```

Figure 1.2 Code piece of calculation of vanilla call option - part of function definitions and price formulas

Answer 2

```
In [23]: # other parameters are declared above
gamma = 0.75
dt = 1/12

def share_price_path(t, N):
    n = int(t/dt)
    Z = norm.rvs(size =[N, n])
    price_path = np.array([[np.float64(S0)]*(n+1)]*N)
    for i in range(n):
        vol = sigma*price_path[:,i]**(gamma-1)
        power = (r-vol**2/2)*dt+vol*np.sqrt(dt)*Z[:,i]
        price_path[:,i+1]=price_path[:,i]*np.exp(power)
    return price_path
```

```
In [24]: # defining the seed
np.random.seed(10)

# sample price path with various sample size
share_price_T = [None]*50
vol_share = [None]*50

for i in range(1,51):
    samples = share_price_path(T, i*1000)
    share_price_T[i-1] = np.mean(samples[:,-1])
    vol_share[i-1] = np.std(samples[:, -1])/np.sqrt(i*1000)
```

Figure 2.1 Code piece of calculation of share price paths using CEV model

- All the sample paths for each case are defined in sample variables. The size of the sample variable for each case is an array of [N x 12]
- And the mean value is calculated for each case of N using the above samples to get the price at T and stored in the share_price_T variable

```
In [25]: plt.plot(np.array(range(1,51))*1000, share_price_T, label="Share Price at T")
         plt.xlabel('Sample Size')
         plt.ylabel('Share Price at T')
         plt.legend()
         plt.show()
```

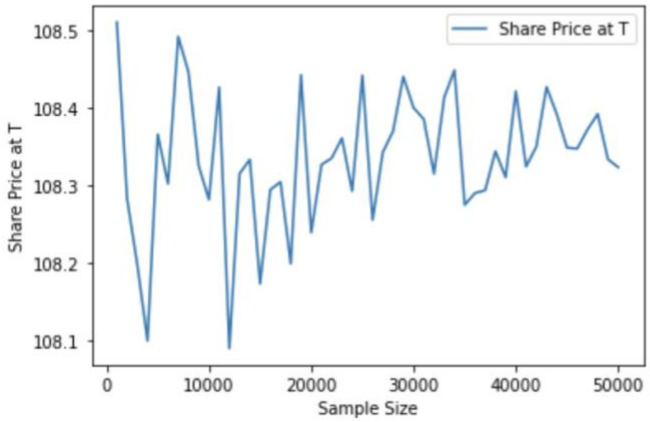

Figure 2.2 Plot showing the different sample sizes and the respective share price at time T

```
In [35]: # visualising 1000 share price paths as an example

         paths_1000 = share_price_path(1, 1000)

         for array in paths_1000:
             plt.plot(array)

         plt.xlabel('Time (months)')
         plt.ylabel('Share price')
         plt.show()
```

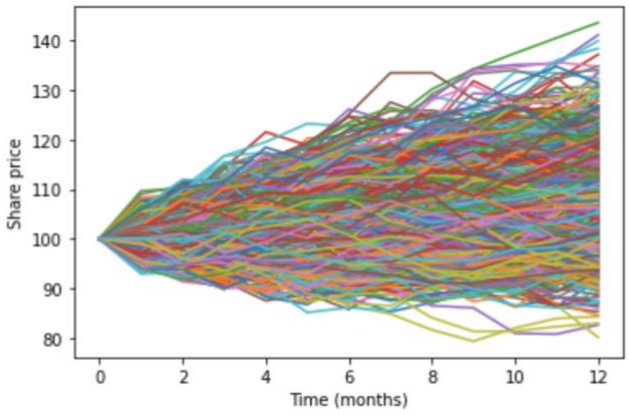

Figure 2.3 Plot showing the visualization of all the share price paths over a year at each month

Answer 3

In [27]:
```
# Set the random seed as above to reproduce the result
np.random.seed(10)

# use the above defined CEV model function.
def call_price_and_stddev(t, N):
    sample = share_price_path(t, N)
    pay_off = np.maximum(sample[:, -1]-K, 0)*np.exp(-r*t)
    return np.mean(pay_off), np.std(pay_off)/np.sqrt(N)
```

In [28]:
```
#Price calculation
call_price = [None]*50
call_stddev = [None]*50

# price estimates
for i in range(1, 51):
    call_price[i-1], call_stddev[i-1] = call_price_and_stddev(T, i*1000)
```

In [29]:
```
# Call price under closed form and the method is CEV
z = 2+1/(1-gamma)
def closed_form_call_price(t):
    kappa = 2*r/(sigma**2*(1-gamma)*(np.exp(2*r*(1-gamma)*t)-1))
    x = kappa*S0**(2*(1-gamma))*np.exp(2*r*(1-gamma)*t)
    y = kappa*K**(2*(1-gamma))
    return S0*(1-ncx2.cdf(y,z,x))-K*np.exp(-r*t)*ncx2.cdf(x,z-2,y)
```

Figure 3.1 code calculation of simulation of call option price using Monte Carlo estimation

The above calculation results are visualized in the following part.

Answer 4

In [30]:
```
# plots - call option price od estimates and in clsoed -form CEV, and error bounds
plt.plot(np.array(range(1,51))*1000, call_price, '.', label="Monte Carlo estimates")
plt.plot(np.array(range(1,51))*1000, [closed_form_call_price(T)]*50, label="Price in closed form")
plt.plot(np.array(range(1,51))*1000, [closed_form_call_price(T)+3*s for s in call_stddev])
plt.plot(np.array(range(1,51))*1000, [closed_form_call_price(T)-3*s for s in call_stddev])
plt.xlabel("Sample Size")
plt.ylabel("Call Price")
plt.legend()
plt.show()
```

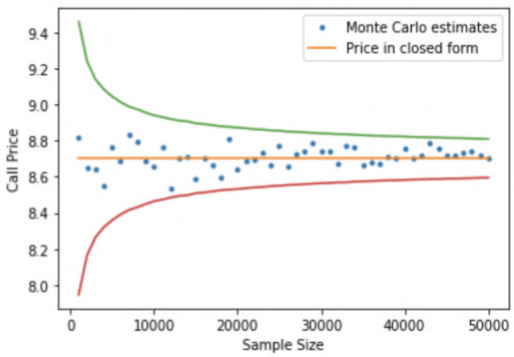

Figure 4.1 Plot showing the visualization of Monte Carlo estimate price and closed form price, and the upper and lower bounds of standard deviation

```
In [31]: # Distribution graph of call_price and estimated share price
         plt.plot(call_price, share_price_T, 'x')
         plt.xlabel("Call Price")
         plt.ylabel("Share Price")
         plt.title("Distribution of call price and share price")
         plt.show()
```

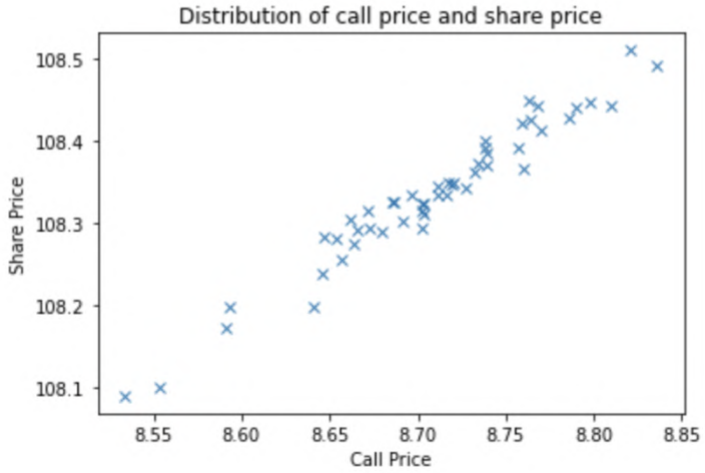

Figure 4.2 Plot showing the distribution of call price and the Share price

Correlation of the above mentioned call price and the share price is calculated and it is **0.9760**

```
In [32]: corr, _ = pearsonr(call_price, share_price_T)

In [33]: corr
Out[33]: 0.976096807863086
```

Figure 4.3 Calculation of Pearson correlation coefficient

Answer 5

Step 1: We take the closing price of USD 324.76 as on October 15, 2021 as the current price. The strike closest to this current price for which call option contracts are available is USD 325

Figure 5.1: Facebook Closing Stock Price on October 15, 2021 (Source: Yahoo Finance)

Step 2: For the strike of USD 325, the below table presents the implied volatilities for call option contracts across a range of maturities.

Expire Date	Contract Name	Implied Volatility
2021-10-22	FB211022C00325000	0.3352
2021-10-29	FB211029C00325000	0.4465
2021-11-05	FB211105C00325000	0.3914
2021-11-12	FB211112C00325000	0.3666
2021-11-19	FB211119C00325000	0.3431
2021-11-26	FB211126C00325000	0.3275
2021-12-17	FB211217C00325000	0.3169
2022-01-21	FB220121C00325000	0.3089
2022-02-18	FB220218C00325000	0.3263
2022-03-18	FB220318C00325000	0.3225
2022-06-17	FB220617C00325000	0.3295
2022-09-16	FB220916C00325000	0.3345
2023-01-20	FB230120C00325000	0.3493
2023-06-16	FB230616C00325000	0.3509
2024-01-19	FB240119C00325000	0.3453

Figure 5.2: Implied Volatilities for Facebook Call Options with K = 325 for Various Maturities (Source: Yahoo Finance)

Step 3: Per Yahoo Finance, three closest strike below USD 325 for which call option contracts across a *wide range of maturities* are traded are USD 320, USD 315, and USD 310. The ones above are USD 330, USD 335 and USD 340. The implied volatilities for these are shown below.

Expire Date	Contract Name	Implied Volatility
2021-10-22	FB211022C00320000	0.3494
2021-10-29	FB211029C00320000	0.4515
2021-11-05	FB211105C00320000	0.3960
2021-11-12	FB211112C00320000	0.3712
2021-11-19	FB211119C00320000	0.3476
2021-11-26	FB211126C00320000	0.3376
2021-12-17	FB211217C00320000	0.3212
2022-01-21	FB220121C00320000	0.3138
2022-02-18	FB220218C00320000	0.3299
2022-03-18	FB220318C00320000	0.3260
2022-04-14	FB220414C00320000	0.3251
2022-06-17	FB220617C00320000	0.3330
2022-09-16	FB220916C00320000	0.3368
2023-01-20	FB230120C00320000	0.3518
2023-03-17	FB230317C00320000	0.3479
2023-06-16	FB230616C00320000	0.3476
2024-01-19	FB240119C00320000	0.3439

K = 320

Expire Date	Contract Name	Implied Volatility
2021-10-22	FB211022C00315000	0.3629
2021-10-29	FB211029C00315000	0.4564
2021-11-05	FB211105C00315000	0.4016
2021-11-12	FB211112C00315000	0.3774
2021-11-19	FB211119C00315000	0.3538
2021-11-26	FB211126C00315000	0.3416
2021-12-17	FB211217C00315000	0.3259
2022-01-21	FB220121C00315000	0.3181
2022-02-18	FB220218C00315000	0.3334
2022-03-18	FB220318C00315000	0.3297
2022-06-17	FB220617C00315000	0.3360
2022-09-16	FB220916C00315000	0.3395
2023-01-20	FB230120C00315000	0.3488
2023-06-16	FB230616C00315000	0.3513
2024-01-19	FB240119C00315000	0.3527

K = 315

Expire Date	Contract Name	Implied Volatility
2021-10-22	FB211022C00310000	0.3836
2021-10-29	FB211029C00310000	0.4644
2021-11-05	FB211105C00310000	0.4090
2021-11-12	FB211112C00310000	0.3844
2021-11-19	FB211119C00310000	0.3596
2021-11-26	FB211126C00310000	0.3480
2021-12-17	FB211217C00310000	0.3313
2022-01-21	FB220121C00310000	0.3229
2022-02-18	FB220218C00310000	0.3384
2022-03-18	FB220318C00310000	0.3335
2022-04-14	FB220414C00310000	0.3323
2022-06-17	FB220617C00310000	0.3389
2022-09-16	FB220916C00310000	0.3424
2023-01-20	FB230120C00310000	0.3513
2023-03-17	FB230317C00310000	0.3539
2023-06-16	FB230616C00310000	0.3525
2024-01-19	FB240119C00310000	0.3550

K = 310

Figure 5.3: Implied Volatilities for Facebook Calls with K = 320, 315, 310 for Various Maturities (Source: Yahoo Finance)

Expire Date	Contract Name	Implied Volatility
2021-10-22	FB211022C00330000	0.3302
2021-10-29	FB211029C00330000	0.4411
2021-11-05	FB211105C00330000	0.3856
2021-11-12	FB211112C00330000	0.3619
2021-11-19	FB211119C00330000	0.3377
2021-11-26	FB211126C00330000	0.3284
2021-12-17	FB211217C00330000	0.3112
2022-01-21	FB220121C00330000	0.3049
2022-02-18	FB220218C00330000	0.3227
2022-03-18	FB220318C00330000	0.3191
2022-04-14	FB220414C00330000	0.3184
2022-06-17	FB220617C00330000	0.3273
2022-09-16	FB220916C00330000	0.3315
2023-01-20	FB230120C00330000	0.3383
2023-03-17	FB230317C00330000	0.3500
2023-06-16	FB230616C00330000	0.3428
2024-01-19	FB240119C00330000	0.3506

K = 330

Expire Date	Contract Name	Implied Volatility
2021-10-22	FB211022C00335000	0.3275
2021-10-29	FB211029C00335000	0.4352
2021-11-05	FB211105C00335000	0.3804
2021-11-12	FB211112C00335000	0.3555
2021-11-19	FB211119C00335000	0.3325
2021-11-26	FB211126C00335000	0.3229
2021-12-17	FB211217C00335000	0.3085
2022-01-21	FB220121C00335000	0.3019
2022-02-18	FB220218C00335000	0.3196
2022-03-18	FB220318C00335000	0.3164
2022-06-17	FB220617C00335000	0.3251
2022-09-16	FB220916C00335000	0.3298
2023-01-20	FB230120C00335000	0.3390
2023-06-16	FB230616C00335000	0.3428
2024-01-19	FB240119C00335000	0.3467

K = 335

Expire Date	Contract Name	Implied Volatility
2021-10-22	FB211022C00340000	0.3272
2021-10-29	FB211029C00340000	0.4319
2021-11-05	FB211105C00340000	0.3762
2021-11-12	FB211112C00340000	0.3505
2021-11-19	FB211119C00340000	0.3275
2021-11-26	FB211126C00340000	0.3194
2021-12-17	FB211217C00340000	0.3043
2022-01-21	FB220121C00340000	0.2983
2022-02-18	FB220218C00340000	0.3171
2022-03-18	FB220318C00340000	0.3137
2022-04-14	FB220414C00340000	0.3130
2022-06-17	FB220617C00340000	0.3223
2022-09-16	FB220916C00340000	0.3278
2023-01-20	FB230120C00340000	0.3368
2023-03-17	FB230317C00340000	0.3457
2023-06-16	FB230616C00340000	0.3416
2024-01-19	FB240119C00340000	0.3425

K = 340

Figure 5.4: Implied Volatilities for Facebook Calls with K = 330, 335, 340 for Various Maturities (Source: Yahoo Finance)

Step 4: We now plot the Call Option Volatility Smile for contracts expiring on 22 October 2021. As expected, the implied volatility for options deep into (K = 330, 335, 340) or out of the money (K = 320, 315, 310) are greater than or equal to the implied volatility when the contracts at the money (K = 325).

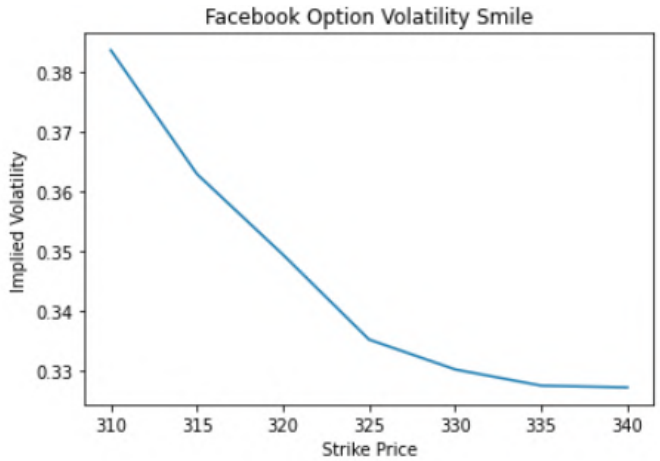

Figure 5.5: Facebook Option Volatility Smile for Calls Expiring on 22 October 2021

Answer 6

Implied volatility refers to the level of volatility which, when plugged into the Black-Scholes pricing equation, results in an analytical price identical to the prevailing market price for an option contract. We use the Newton-Raphson Algorithm, which is an iterative approach for finding approximations to the roots of a real-valued function, to estimate implied volatility.

Basic Approach:
Given the Black-Scholes option pricing equation:

$$c = S_0 N(d_1) - Ee^{-rt} N(d_2)$$

where
c: Theoretical call price
S_0: Current stock price
$d_1 = \frac{\ln\left(\frac{S_0}{E}\right) + \left(r + \frac{1}{2}\sigma^2\right)t}{\sigma\sqrt{t}}$ and $d_2 = \frac{\ln\left(\frac{S_0}{E}\right) + \left(r - \frac{1}{2}\sigma^2\right)t}{\sigma\sqrt{t}} = d_1 - \sigma\sqrt{t}$

E: Strike price
$N(.)$: Cumulative standard normal distribution

We first express volatility as a function of the above equation, as shown below:
$f(\sigma) = S_0 N(d_1) - Ee^{-rt} N(d_2) - c = 0$

The algorithm to estimate implied volatility for *n* iterations is as follows:

$$\sigma_{n+1} = \sigma_n - \frac{f(\sigma_n)}{f'(\sigma_n)}$$

STOP when $|\sigma_{n+1} - \sigma_n|$ is small

where $f'(\sigma)$ is the derivative of the function w.r.t volatility. It is given by:

$$f'(\sigma) = S_0 f(d_1) d_1' - \frac{E}{e^{rt}} f(d_2) d_2'$$

The expressions for the derivatives d_1' and d_2' with respect to volatility are as follows:

$$d_1' = \frac{\sigma^2 t\sqrt{t} - \left\{\ln\left(\frac{S_0}{E}\right) + \left(r + \frac{\sigma^2}{2}\right)t\right\}\sqrt{t}}{\sigma^2 t}$$

$$d_2' = d_1' - \sqrt{t}$$

Code
Step-1: We use the following inputs to run the algorithm. We assume a smallness threshold of 0.00000001, 100 iterations and an initial volatility of 10%

```
S0 = 324.76 #Facebook stock price at October 15, 2021 close
K = 325
```

```
#Inputs
r_f = 0.04 #1-month T-bill rate as on October 15
#Source: https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/TextView.aspx?data=billrates
v_0 = 0.1
threshold = 0.00000001
iterations = 100
```

Step-2: We define the functions listed above.

```
#Functions
def d1(current_stock_price, strike, risk_free_rate, volatility, time_to_maturity):

    return (np.log(current_stock_price/strike) + (risk_free_rate + volatility**2/2)*time_to_maturity)/(
        volatility*np.sqrt(time_to_maturity))

def d2(d_1, volatility, time_to_maturity):
    return d_1 - volatility * np.sqrt(time_to_maturity)

def f_sigma(current_stock_price, d_1, d_2, strike, risk_free_rate, time_to_maturity, call_market_price):

    return (current_stock_price * stats.norm.cdf(d_1) - strike * np.exp(-risk_free_rate*time_to_maturity)
            * stats.norm.cdf(d_2) - call_market_price)

def Dd1_Dsigma(volatility, time_to_maturity, current_stock_price, strike, risk_free_rate):

    return (volatility**2*time_to_maturity*np.sqrt(time_to_maturity)-
            (np.log(current_stock_price/strike)+(risk_free_rate+volatility**2/2)*time_to_maturity)
            *np.sqrt(time_to_maturity))/(volatility**2*time_to_maturity)

def Dd2_Dsigma(d1_derivative, time_to_maturity):
    return d1_derivative - np.sqrt(time_to_maturity)

def Df_sigma_Dsigma(current_stock_price, d_1, d1_derivative, strike, risk_free_rate, time_to_maturity, d_2,
                    d2_derivative):

    return (current_stock_price*stats.norm.pdf(d_1)*d1_derivative
            - strike*np.exp(-risk_free_rate*time_to_maturity)*stats.norm.pdf(d_2)*d2_derivative)
```

Step-3: Based on the above functions, the implied volatility function is written as follows:

```
#Algorithm
def implied_volatility_function(call_price, time_to_maturity, strike):
    vol_sequence = []
    vol_sequence.append(v_0)

    for i in range(0,iterations):
        d_1 = d1(S0, strike, r_f, vol_sequence[i], time_to_maturity)
        d_2 = d2(d_1, vol_sequence[i], time_to_maturity)

        d1_derivative = Dd1_Dsigma(vol_sequence[i], time_to_maturity, S0, strike, r_f)
        d2_derivative = Dd2_Dsigma(d1_derivative, time_to_maturity)

        vol_sequence.append(vol_sequence[i] - f_sigma(S0, d_1, d_2, strike, r_f, time_to_maturity, call_price)/
                            Df_sigma_Dsigma(S0, d_1, d1_derivative, strike, r_f, time_to_maturity, d_2, d2_derivative)

        if abs(vol_sequence[i+1] - vol_sequence[i]) < threshold: break

    return vol_sequence[-1]
```

Validating Implied Volatilities for 2 Option Contracts

We estimate the implied volatility and compare it with the one reported on Yahoo finance for two contracts:
325 Call expiring on 19 Nov 2021 - FB211119C00325000
325 Call expiring on 17 Dec 2021 - FB211217C00325000

Looking at the results below, the estimate from the bisection algorithm seems pretty accurate.

325 Call 19 Nov 2021 - FB211119C00325000

```
vol_bisection = implied_volatility_function(13.30, 1/12, 325)
print('Implied Volatility Based on Algorithm:', "{:.2%}".format(vol_bisection))
print('Reported Implied Volatility: 34.31%')
print('Difference:', "{:.2%}".format(0.3431-vol_bisection))
```

```
Implied Volatility Based on Algorithm: 34.48%
Reported Implied Volatility: 34.31%
Difference: -0.17%
```

325 Call 17 Dec 2021 - FB211217C00325000

```
vol_bisection = implied_volatility_function(16.70, 2/12, 325)
print('Implied Volatility Based on Algorithm:', "{:.2%}".format(vol_bisection))
print('Reported Implied Volatility: 31.69%')
print('Difference:', "{:.2%}".format(0.3169-vol_bisection))
```

```
Implied Volatility Based on Algorithm: 29.83%
Reported Implied Volatility: 31.69%
Difference: 1.86%
```

Answer 7

The code to estimate volatility skewness (essentially the slope of the volatility smile) for strike levels below K = 325 for the call expiring on October 22, 2021 and the resulting output are shown below.

```
df_vol_skewness = pd.DataFrame(list(zip(strikes, vol_2021_10_22)), columns = ['Strike Price', 'Implied Volatility'])
df_vol_skewness['Delta Strike Price'] = df_vol_skewness['Strike Price'].diff()
df_vol_skewness['Delta Implied Volatility'] = df_vol_skewness['Implied Volatility'].diff()
df_vol_skewness['Volatility Skewness'] = (df_vol_skewness['Delta Implied Volatility']
                                          / df_vol_skewness['Delta Strike Price'])

df_vol_skewness.drop([4, 5, 6])
```

	Strike Price	Implied Volatility	Delta Strike Price	Delta Implied Volatility	Volatility Skewness
0	310	0.3836	NaN	NaN	NaN
1	315	0.3629	5.0	-0.0207	-0.00414
2	320	0.3494	5.0	-0.0135	-0.00270
3	325	0.3352	5.0	-0.0142	-0.00284

Answer 8

Volatility does not depend on the strike level under the Black-Scholes model. This is because the model assumes that the underlying stock's price follows a Geometric Brownian motion with constant volatility. So, according to the model, when we price a European option, we will plug in the same volatility parameter into the pricing equation irrespective of the option's strike level or time to maturity.

In practice, however, prices observed in options markets (which are well-developed) suggest that volatility is not constant across varying strike levels and times to maturity. Concretely, if one plugs in the observed market prices of options contracts with the same time to maturity into the Black-Scholes equation along with their respective strike levels, the implied volatilities which satisfy the pricing equation vary depending on strike levels. They are lower for options at-the-money relative to options deep in-the-money or out-of-the-money (implied volatility smile).

Answer 9

The Heston model better estimates the volatility smile by producing results regarding volatility which tend to be in line with market observations, i.e. volatility is stochastic, non-negative and mean-reverting. In addition, the Heston model also provides a quasi-closed form solution for European options. In contrast, the Black-Scholes model provides a closed-form solution under the unrealistic assumption that volatility is constant w.r.t strike levels.

Under the Black-Scholes model, the underlying has the following dynamics with constant volatility:

$$dS_t = \mu S_t dt + \sigma S_t dW_t, S_0 = S$$

In contrast, the Heston model prescribes the following dynamics:

$$dS_t = \mu S dt + \sqrt{v_t} S dW_t^1$$

The volatility term in the equation above, v_t, follows a CIR, or square-root process:

$$d\sqrt{v_t} = -\beta\sqrt{v_t} dt + \sigma dW_t^2$$

After applying Ito's formula to the above volatility process and with some manipulation, this can be written as:

$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dW_t^2$$

Answer 10

The Black-Scholes-Merton (BSM) model for option pricing was developed in the 1970s, following which it gained a lot of popularity. This is because the model provides a simple equation to calculate the theoretical price or fair value of an option, by plugging in inputs observed in the market. The BSM model requires five parameters to estimate an option contract's price, viz. the underlying's spot price, the option's strike and time to maturity, the risk-free interest rate and the implied volatility. Of these, only implied volatility is not directly observable in the market. Consequently, as options markets matured, practitioners started solving for implied volatility based on the other 4 parameters and the option's market price.

After the market crash of 1987, such implied volatilities started displaying a U-shaped relationship with strike prices for options with similar times to maturity. The phenomenon came to be known as the volatility smile. It demonstrated that the constant volatility assumption underpinning the BSM framework was no longer realistic. In reality, volatility typically has a negative correlation with stock price, implying that it tends to increase when prices decline and decrease when they increase. This is called the leverage effect. Volatility also exhibits what is called the clustering effect, which means that larger market moves are followed by larger moves. Such a characterisation of volatility clearly cannot be accounted for by the BSM.

Besides this, another unrealistic assumption on which BSM is based is that of lognormal stock prices. This is because, in reality, stock return distributions are far from normal and typically have fatter tails. Since the two assumptions do not characterise option markets, alternative approaches are needed to price options.

One of the popular models which attempts to address the above shortcomings of the BSM framework is the Heston model introduced in 1993 by Steven Heston. Unlike BSM, the model does not require stock prices to be lognormal or volatility to be constant. Instead, it accounts for the actual distribution of stock prices and the phenomenon of volatility smile into account when pricing options.

The Heston model postulates that both the underlying stock price and variance (which is assumed to be mean-reverting) follow Brownian motions and that these two are correlated. It is based on nine input parameters, which are stock price, strike price K, the risk-free interest rate, maturity, initial volatility, long-term volatility, the mean-reverting speed for volatility, and the correlation between stock price and volatility.

The following indicates how the Heston model's parameters capture the real-world phenomena exhibited by volatility (volatility smile, leverage effect, clustering effect)

- The initial variance and long-term variance parameters allows adjustment of the height of the volatility smile
- Mean reversion speed controls the degree of volatility clustering, and the curvature of the volatility smile
- The volatility of volatility parameter controls the kurtosis of the underlying stock price's distribution
- The correlation between returns and volatility controls the skewness of the distribution

Besides allowing for a correlation between volatility and stock price, another advantage of the Heston model is that it provides a quasi-closed form solution for pricing European options compared to other stochastic volatility models. Specifically, the model provides a closed-form expression for the underlying's characteristic function which can then be used with Fourier pricing techniques to estimate option prices. This also yields the model's parameters to easy calibration with respect to market data. Lastly, in terms of implementation, the Heston model is computationally cheaper than models which require Monte Carlo simulations.

To conclude, the Heston model is superior to BSM due to its ability to capture several phenomena (volatility smile, leverage and clustering effects) which the underlying stock's volatility exhibits in markets. And relative to other stochastic volatility models, it is preferred for its quasi-closed form solution and its computational efficiency.