MF 796: Computational Methods of Mathematical Finance

Professors Christopher Kelliher and Eugene Sorets Spring 2024

Problem set 3

1. Implementation of Breeden-Litzenberger:

You are given the following volatility data, where the table of strikes is quoted in terms of Deltas. The DP rows indicate "X Delta Puts" and the DC rows indicate "X Delta Calls". You also know that the current stock price is 100, the risk-free rate is 0, and the asset pays no dividends.

Expiry / Strike	1M	3M
10DP	32.25%	28.36%
25DP	24.73%	21.78%
40DP	20.21%	18.18%
50D	18.24%	16.45%
40DC	15.74%	14.62%
25DC	13.70%	12.56%
10DC	11.48%	10.94%

NOTE: The table of strikes is quoted in terms of Deltas, where the DP rows indicate ″X Delta Puts″ and the DC rows indicate ″X Delta Calls″

(a) Using the table of quoted (Black-Scholes) Deltas and volatilities, extract a table of strikes corresponding to each option.

In [26]:

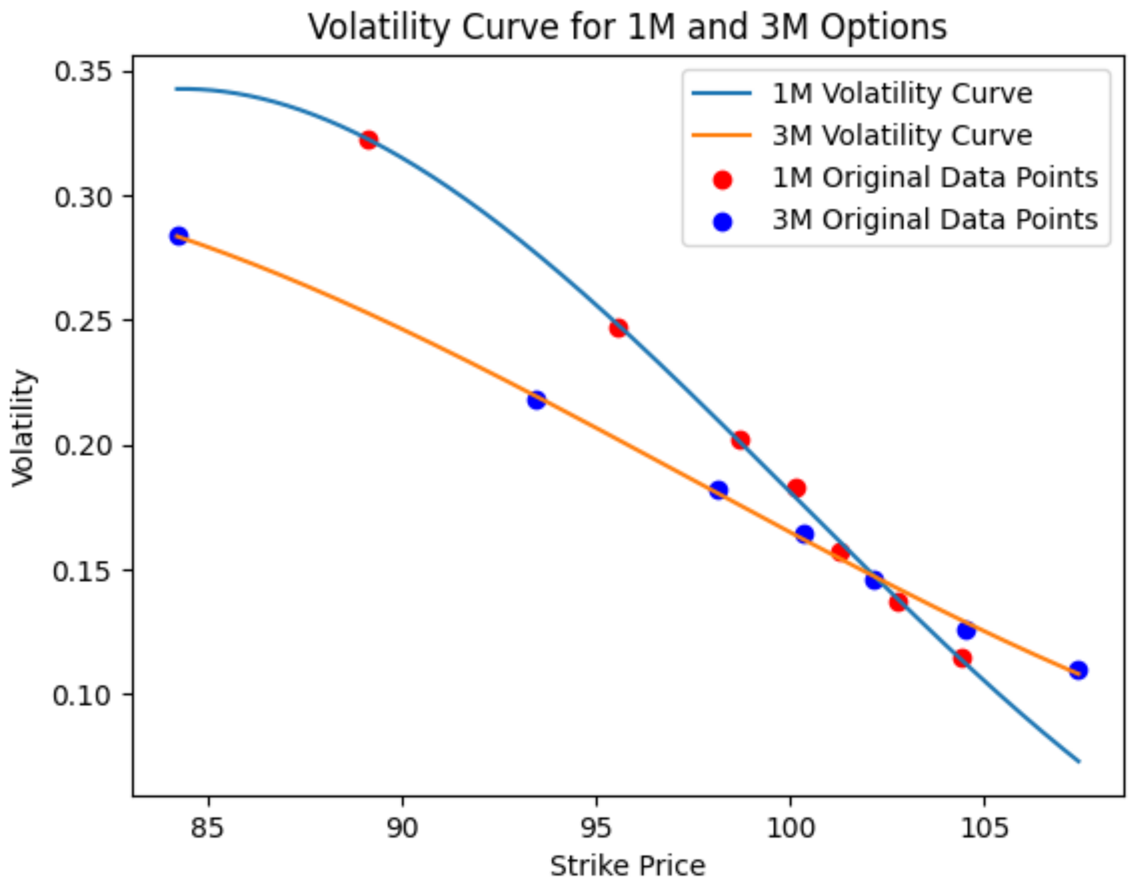
```
import numpy as np
from scipy.stats import norm
from scipy.optimize import brentq

# Current stock price, risk-free rate, and the underlying asset does not pay
S = 100
r = 0.0

def d1(S, K, T, r, sigma):
    return (np.log(S/K) + (r + sigma**2 / 2) * T) / (sigma * np.sqrt(T))

def call_delta(S, K, T, r, sigma):
    return norm.cdf(d1(S, K, T, r, sigma))

def put_delta(S, K, T, r, sigma):
    return -norm.cdf(-d1(S, K, T, r, sigma))

# Option data: Delta values and volatilities
options_data = [
    ('10DP', 0.10, 0.3225, 0.2836, 'put'),
    ('25DP', 0.25, 0.2473, 0.2178, 'put'),
    ('40DP', 0.40, 0.2021, 0.1818, 'put'),
    ('50D', 0.50, 0.1824, 0.1645, 'call'),
    ('40DC', 0.40, 0.1574, 0.1462, 'call'),
    ('25DC', 0.25, 0.1370, 0.1256, 'call'),
    ('10DC', 0.10, 0.1148, 0.1094, 'call'),
]

def solve_for_strike(delta, sigma, option_type, T=1/12):
    if option_type == 'put':
        delta = -delta  # Convert delta to negative for put options
    func = lambda K: (call_delta(S, K, T, r, sigma) if option_type == 'call'
    return brentq(func, 0.01 * S, 2 * S)

# Calculate and print the strike price for each option
print("1 Month Strikes:")
for label, delta, sigma_1m, sigma_3m, option_type in options_data:
    K_1m = solve_for_strike(delta, sigma_1m, option_type, T=1/12)
    print(f"{label}: 1M Strike Price = {K_1m:.2f}")

print("\n3 Months Strikes:")
for label, delta, sigma_1m, sigma_3m, option_type in options_data:
    K_3m = solve_for_strike(delta, sigma_3m, option_type, T=3/12)
    print(f"{label}: 3M Strike Price = {K_3m:.2f}")
```

```
1 Month Strikes:
10DP: 1M Strike Price = 89.14
25DP: 1M Strike Price = 95.54
40DP: 1M Strike Price = 98.70
50D: 1M Strike Price = 100.14
40DC: 1M Strike Price = 101.26
25DC: 1M Strike Price = 102.78
10DC: 1M Strike Price = 104.40

3 Months Strikes:
10DP: 3M Strike Price = 84.23
25DP: 3M Strike Price = 93.47
40DP: 3M Strike Price = 98.13
50D: 3M Strike Price = 100.34
40DC: 3M Strike Price = 102.14
25DC: 3M Strike Price = 104.53
10DC: 3M Strike Price = 107.42
```

(b) Choose an interpolation scheme that defines the volatility function for all strikes, $\sigma(K)$.

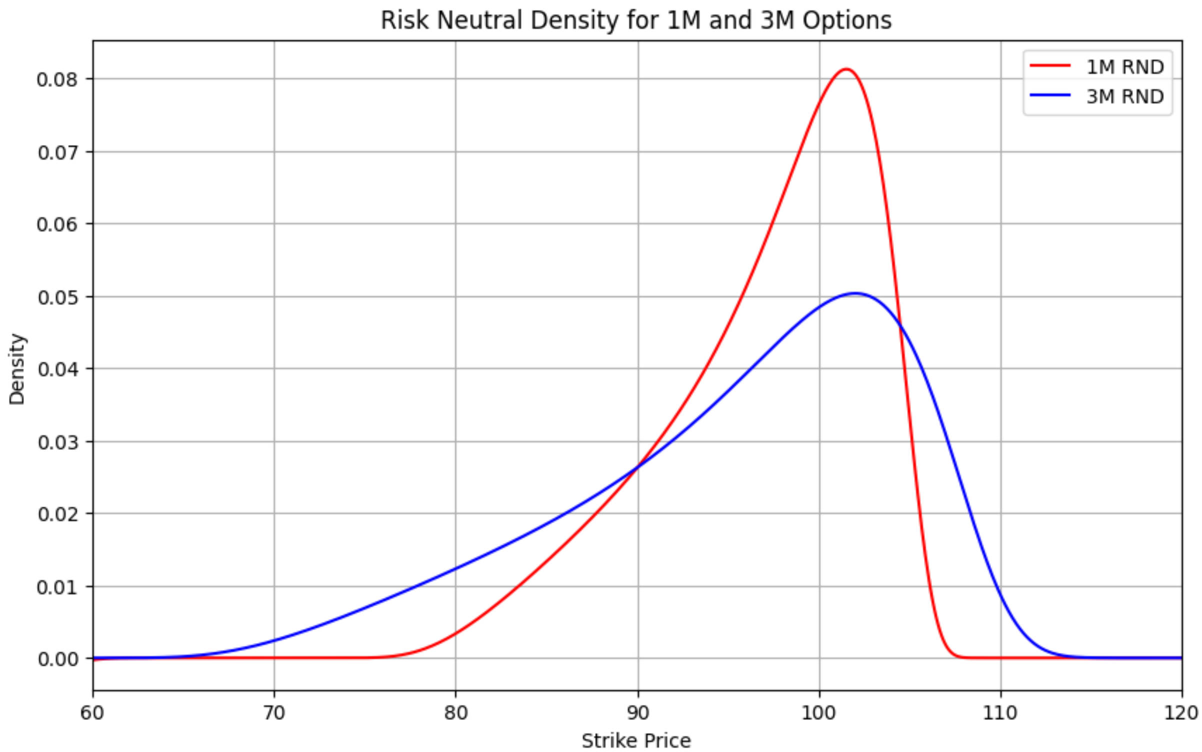
In [27]:

```
import matplotlib.pyplot as plt

# Initialize lists to collect strike prices (K) and volatilities (sigma) for
strikes_1m = []
volatilities_1m = []
strikes_3m = []
volatilities_3m = []

# Calculate strike prices and volatilities for 1M and 3M options
for label, delta, sigma_1m, sigma_3m, option_type in options_data:
    K_1m = solve_for_strike(delta, sigma_1m, option_type, T=1/12)
    strikes_1m.append(K_1m)
    volatilities_1m.append(sigma_1m)

    K_3m = solve_for_strike(delta, sigma_3m, option_type, T=3/12)
    strikes_3m.append(K_3m)
    volatilities_3m.append(sigma_3m)

# Convert lists to NumPy arrays for convenience in interpolation
strikes_1m = np.array(strikes_1m)
volatilities_1m = np.array(volatilities_1m)
strikes_3m = np.array(strikes_3m)
volatilities_3m = np.array(volatilities_3m)

# Apply polynomial interpolation, assuming a 3rd degree polynomial here
degree = 3
poly_coeffs_1m = np.polyfit(strikes_1m, volatilities_1m, degree)
poly_coeffs_3m = np.polyfit(strikes_3m, volatilities_3m, degree)

# Generate polynomial functions
poly_func_1m = np.poly1d(poly_coeffs_1m)
poly_func_3m = np.poly1d(poly_coeffs_3m)

# Generate dense strike price points for plotting
K_dense = np.linspace(min(strikes_1m.min(), strikes_3m.min()), max(strikes_1

# Calculate volatilities at these strike price points for 1M and 3M
vol_dense_1m = poly_func_1m(K_dense)
vol_dense_3m = poly_func_3m(K_dense)

# Plot volatility curves for 1M and 3M
plt.plot(K_dense, vol_dense_1m, label='1M Volatility Curve')
plt.plot(K_dense, vol_dense_3m, label='3M Volatility Curve')
plt.scatter(strikes_1m, volatilities_1m, color='red', label='1M Original Dat
plt.scatter(strikes_3m, volatilities_3m, color='blue', label='3M Original Da
plt.xlabel('Strike Price')
plt.ylabel('Volatility')
plt.title('Volatility Curve for 1M and 3M Options')
plt.legend()
plt.show()
```



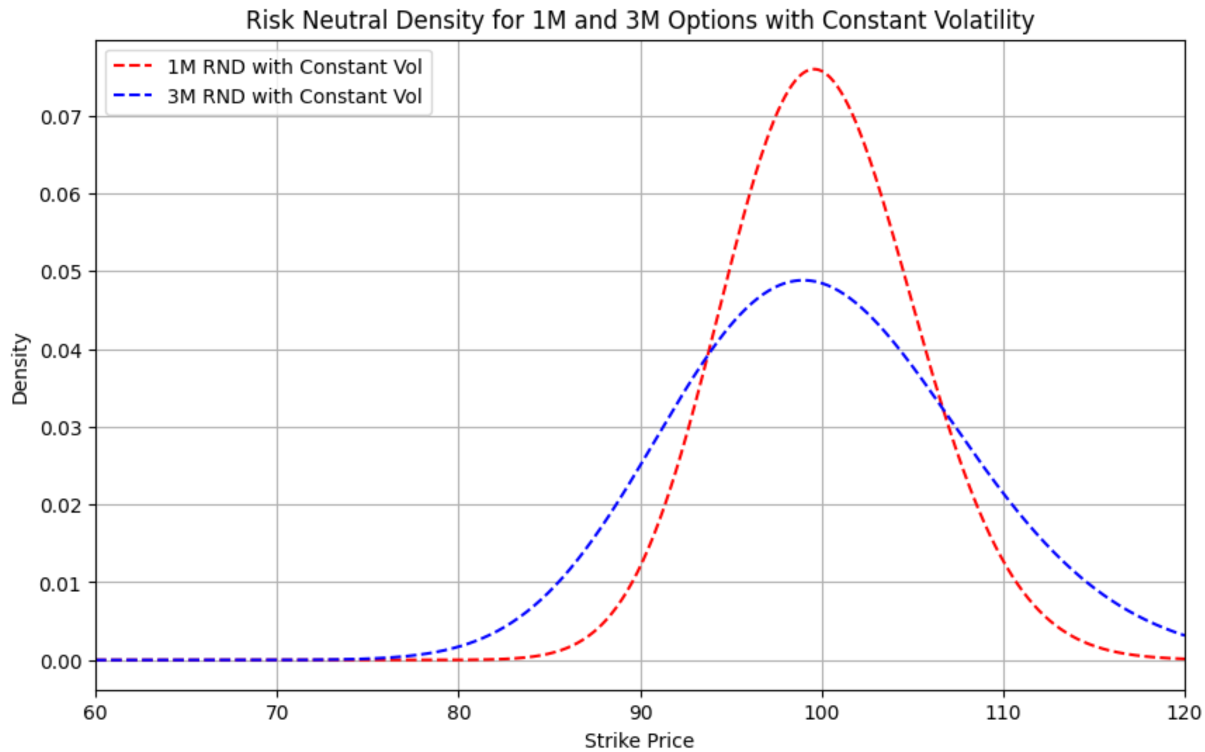
(c) Extract the risk neutral density for 1 & 3 month options. Comment on the differences between the two distributions. Is it what you would expect?

In [30]:
```
def blackscholes(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    price = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return price

def calculate_rnd(vol_curve, strikes, T, S=S, r=r):
    h = 0.01
    density = []

    for K in strikes:
        sigma = vol_curve(K)
        price_plus_h = blackscholes(S, K+h, T, r, sigma)
        price = blackscholes(S, K, T, r, sigma)
        price_minus_h = blackscholes(S, K-h, T, r, sigma)

        second_derivative = (price_plus_h - 2*price + price_minus_h) / h**2
        density.append(second_derivative)

    return np.array(density)

K_dense = np.linspace(60, 120, 600)

density_1m = calculate_rnd(poly_func_1m, K_dense, 1/12)
density_3m = calculate_rnd(poly_func_3m, K_dense, 3/12)

plt.figure(figsize=(10, 6))
plt.plot(K_dense, density_1m, label='1M RND', color='red')
plt.plot(K_dense, density_3m, label='3M RND', color='blue')
plt.xlabel('Strike Price')
plt.ylabel('Density')
plt.title('Risk Neutral Density for 1M and 3M Options')
plt.xlim(60, 120)
plt.legend()
plt.grid(True)
plt.show()
```



(d) Extract the risk neutral density for 1 & 3 month options using a constant volatility equal to the 50D volatility. Contrast these densities to the densities obtained above.

In [31]:
```
sigma_50d_1m = 0.1824
sigma_50d_3m = 0.1645


def calculate_rnd_constant_vol(K_dense, sigma, T, S=100, r=0):
    density = []
    h = 0.01
    for K in K_dense:
        price_plus_h = blackscholes(S, K+h, T, r, sigma)
        price = blackscholes(S, K, T, r, sigma)
        price_minus_h = blackscholes(S, K-h, T, r, sigma)
        second_derivative = (price_plus_h - 2*price + price_minus_h) / h**2
        density.append(second_derivative)
    return np.array(density)

density_1m_const_vol = calculate_rnd_constant_vol(K_dense, sigma_50d_1m, 1/1
density_3m_const_vol = calculate_rnd_constant_vol(K_dense, sigma_50d_3m, 3/1

plt.figure(figsize=(10, 6))
plt.plot(K_dense, density_1m_const_vol, label='1M RND with Constant Vol', co
plt.plot(K_dense, density_3m_const_vol, label='3M RND with Constant Vol', co
plt.xlabel('Strike Price')
plt.ylabel('Density')
plt.title('Risk Neutral Density for 1M and 3M Options with Constant Volatili
plt.xlim(60, 120)
plt.legend()
plt.grid(True)
plt.show()
```



(e) Price the following European Options using the densities you constructed in (1c).

i. 1M European Digital Put Option with Strike 110.

ii. 3M European Digital Call Option with Strike 105.

iii. 2M European Call Option with Strike 100.

In [32]:
```
def price_option(density, strikes, option_type, K):
    price = 0
    h = strikes[1] - strikes[0]

    if option_type == 'digital_put':
        relevant_densities = [density[i] for i, strike in enumerate(strikes)
    elif option_type == 'digital_call':
        relevant_densities = [density[i] for i, strike in enumerate(strikes)
    elif option_type == 'call':
        relevant_densities = [max(0, strikes[i]-K) * density[i] for i in ran

    price = sum(relevant_densities) * h
    return price


strikes = np.arange(65, 120, 0.1)

price_1M_digital_put = price_option(density_1m, strikes, 'digital_put', 110)
price_3M_digital_call = price_option(density_3m, strikes, 'digital_call', 10
price_2M_call = (price_option(density_1m, strikes, 'call', 100) + price_opti

print(f"1M Digital Put Option with Strike 110 Price: {price_1M_digital_put}"
print(f"3M Digital Call Option with Strike 105 Price: {price_3M_digital_call
print(f"2M Call Option with Strike 100 Price: {price_2M_call}")
```

```
1M Digital Put Option with Strike 110 Price: 0.9730495350145911
3M Digital Call Option with Strike 105 Price: 0.3817846528841092
2M Call Option with Strike 100 Price: 3.853684530408705
```

2. Calibration of Heston Model

Recall that the Heston Model is defined by the following system of SDE's:

$$dS_t = rS_t dt + \sqrt{\nu_t} S_t dW_t^1 \tag{1}$$
$$d\nu_t = \kappa(\theta - \nu_t)dt + \sigma\sqrt{\nu_t} dW_t^2 \tag{2}$$
$$\mathrm{Cov}(dW_t^1, dW_t^2) = \rho dt \tag{1}$$

Recall also that the characteristic function for the Heston Model is known to be:

$$\omega(u) = \exp\left[iu \ln S_0 + iu(r-q)t + \frac{\sigma^{-2}\{\kappa\theta t(\kappa - i\rho\sigma u)\}}{\left(\cosh\frac{\lambda t}{2} + \frac{\kappa - i\rho\sigma u}{\lambda}\sinh\frac{\lambda t}{2}\right)^{\frac{2\kappa\theta}{\sigma^2}}}\right] \tag{2}$$

$$\Phi(u) = \omega(u)\exp\left[-\frac{(u^2+iu)\nu_0}{\lambda\coth\frac{\lambda t}{2} + \kappa - i\rho\sigma u}\right] \tag{3}$$

$$\lambda = \sqrt{\sigma^2(u^2+iu) + (\kappa - i\rho\sigma u)^2} \tag{4}$$

See the attached spreadsheet for options data. $r = 1.5\%$, $q = 1.77\%$, $S_0 = 267.15$.

Consider the given market prices and the following equal weight least squares minimization function:

$$\vec{p}_{min} = \min_{\vec{p}}\left\{\sum_{\tau,K}(\tilde{c}(\tau, K, \vec{p}) - c_{\tau,K})^2\right\} \tag{5}$$

where $\tilde{c}(\tau, K, \vec{p})$ is the FFT based model price of a call option with expiry τ and strike K.

(a) Check the option prices for arbitrage. Are there arbitrage opportunities at the mid? How about after accounting for the bid-ask spread? Remove any arbitrage violations from the data.

In [33]:
```python
import pandas as pd

# Load the options data from the provided Excel file
file_path = '/Users/apple/Downloads/mf796-hw3-opt-data.xlsx'
options_data = pd.read_excel(file_path)

# Display the first few rows of the dataset to understand its structure
options_data.head()
```

Out[33]:

	expDays	expT	K	call_bid	call_ask	put_bid	put_ask
0	49	0.134155	240	29.52	29.74	1.52	1.54
1	49	0.134155	245	24.87	25.09	1.89	1.91
2	49	0.134155	250	20.36	20.55	2.38	2.40
3	49	0.134155	255	16.02	16.19	3.03	3.07
4	49	0.134155	260	11.95	12.08	3.94	3.98

In [34]:

```
# Calculate the mid prices for call and put options
options_data['call_mid'] = (options_data['call_bid'] + options_data['call_as
options_data['put_mid'] = (options_data['put_bid'] + options_data['put_ask']

# Check for arbitrage opportunities at the mid price by ensuring monotonic d
# and monotonic increase in put option prices with increasing strike price
arbitrage_free_calls_mid = options_data['call_mid'].diff().dropna() <= 0
arbitrage_free_puts_mid = options_data['put_mid'].diff().dropna() >= 0

# Identifying rows where arbitrage violations occur
violations_calls_mid = arbitrage_free_calls_mid[arbitrage_free_calls_mid ==
violations_puts_mid = arbitrage_free_puts_mid[arbitrage_free_puts_mid == Fal

# Remove identified arbitrage violations from the dataset
options_data_cleaned = options_data.drop(violations_calls_mid.index.union(vi

# Check and adjust for bid-ask spread
# For call options: remove any option where a higher strike has a higher bid
# For put options: remove any option where a lower strike has a higher ask p
call_bid_ask_violations = options_data_cleaned['call_bid'].diff().dropna() >
put_bid_ask_violations = options_data_cleaned['put_ask'].diff().dropna() < 0

# Identifying rows where bid-ask arbitrage violations occur
violations_call_bid_ask = call_bid_ask_violations[call_bid_ask_violations ==
violations_put_bid_ask = put_bid_ask_violations[put_bid_ask_violations == Tr

# Final cleaning by removing identified bid-ask spread violations
final_cleaned_data = options_data_cleaned.drop(violations_call_bid_ask.index

# Summary of actions
summary = {
    'initial_data_count': len(options_data),
    'after_mid_check_count': len(options_data_cleaned),
    'final_data_count': len(final_cleaned_data),
    'removed_for_mid_violations': len(violations_calls_mid) + len(violations
    'removed_for_bid_ask_violations': len(violations_call_bid_ask) + len(vio
}

summary
```

Out[34]:
```
{'initial_data_count': 44,
 'after_mid_check_count': 42,
 'final_data_count': 40,
 'removed_for_mid_violations': 4,
 'removed_for_bid_ask_violations': 4}
```

The initial dataset contained 44 options.

After checking for arbitrage opportunities at the mid-price level and removing violations, 42 options remained.

Considering the bid-ask spread and further removing violations, the dataset was reduced to 40 options.

A total of 4 options were removed due to arbitrage violations at the mid-price level, and another 4 options were removed due to violations after accounting for the bid-ask spread.

(b) Using the FFT Pricing code from a prior homework, find the values of κ, θ, σ, ρ and v0 that minimize the equal weight least squared pricing error. You may choose the starting point and upper and lower bounds of the optimization. You may also choose whether to calibrate to calls, puts, or some combination of the two.

Note that you are given data for multiple expiries, each of which should use the same parameter set, but will require a separate call to the FFT algorithm.

In [35]:
```python
import numpy as np
from scipy import interpolate
import time
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

In [36]:
```python
#Define the characteristic equation of the Heston Model
def heston_characteristic_function(u, params):
    iu = 1j * u

    kappa, theta, sigma, rho, v0, t, S0, r, q  = params['kappa'], params['th

    lambda_ = np.sqrt(sigma**2 * (u**2 + iu) + (kappa - iu * rho * sigma)**2

    omega_numerator =  np.exp(iu * np.log(S0) + iu * (r - q) * t + kappa * t

    omega_denominator  = (np.cosh(lambda_ * t * 0.5) + (kappa - iu * rho * s

    omega_u = omega_numerator/omega_denominator

    coth_lambda_t = 1/np.tanh(0.5 * lambda_ * t)

    phi = omega_u * np.exp(-((u**2 + iu) * v0)/(lambda_ * coth_lambda_t + (k

    return phi
```

In [37]:
```
#Calculate the European Call Option Price of the Heston Model using FFT
def calc_fft_heston_call_prices(alpha, params, N, delta_v, K = None):
    #delta is the indicator function
    kappa, theta, sigma, rho, v0, t, S0, r, q  = params['kappa'], params['th

    begin = time.time()

    delta = np.zeros(N)
    delta[0] = 1
    delta_k = (2*np.pi)/(N*delta_v)

    if K == None:
        #middle strike is at the money
        beta = np.log(S0) - delta_k*N*0.5

    else:
        #middle strike is K
        beta = np.log(K) - delta_k*N*0.5

    k_list = np.array([(beta +(i-1)*delta_k) for i in range(1,N+1) ])
    v_list = np.arange(N) * delta_v

    #building fft input vector
    x_numerator = np.array( [((2-delta[i])*delta_v)*np.exp(-r*t)  for i in r
    x_denominator = np.array( [2 * (alpha + 1j*i) * (alpha + 1j*i + 1) for i
    x_exp = np.array( [np.exp(-1j*(beta)*i) for i in v_list] )

    x_list = (x_numerator/x_denominator)*x_exp* np.array([heston_characteris

    #fft output
    y_list = np.fft.fft(x_list)
    #recovering prices
    prices = np.array( [(1/np.pi) * np.exp(-alpha*(beta +(i-1)*delta_k)) * n

    end = time.time()
    deltatime = end - begin
    return prices, np.exp(k_list), deltatime
```

In [40]:
```
from scipy.optimize import minimize

def optimization_objective(params, market_prices, K, expiries, S0, r, q, N,
    total_error = 0
    for i, expiry in enumerate(expiries):
        params['t'] = expiry
        model_prices, strikes, _ = calc_fft_heston_call_prices(alpha, params
        interp_func = interpolate.interp1d(strikes, model_prices, kind='cubi
        model_price_at_K = interp_func(K[i])
        error = (model_price_at_K - market_prices[i]) ** 2
        total_error += error

    return total_error
```

In [48]:
```
from scipy.optimize import minimize
market_prices = final_cleaned_data['call_mid'].values
K = final_cleaned_data['K'].values
expiries = final_cleaned_data['expT'].values

S0 = 267.15
r = 0.015
q = 0.0177
N = 4096
delta_v = 0.25
alpha = 1.5

initial_params = [1.5, 0.04, 0.2, -0.5, 0.04]  # kappa, theta, sigma, rho, v
bounds = [(0.1, 10), (0.01, 0.5), (0.01, 1), (-0.99, 0.99), (0.01, 0.5)]

def print_status(xk):
    print(f"Current params: {xk}")

def optimization_objective(params, market_prices, K, expiries, S0, r, q, N,
    total_error = 0
    for i, expiry in enumerate(expiries):
        local_params = {'kappa': params[0], 'theta': params[1], 'sigma': par
        model_prices, strikes, _ = calc_fft_heston_call_prices(alpha, local_
        interp_func = interpolate.interp1d(strikes, model_prices, kind='cubi
        model_price_at_K = interp_func(K[i])
        error = (model_price_at_K - market_prices[i]) ** 2
        total_error += error
    print(f"Total error: {total_error}")
    return total_error


result = minimize(optimization_objective, initial_params,
                  args=(market_prices, K, expiries, S0, r, q, N, delta_v, al
                  bounds=bounds,
                  method='L-BFGS-B',
                  options={'disp': True, 'maxiter': 25, 'gtol': 1e-3, 'ftol'
                  callback=print_status)

print("initial_params", initial_params)
print("bounds", bounds)
print("Final optimized parameters:", result.x)
```

```
Total error: 176.38057869328037
Total error: 176.38057880412615
Total error: 176.38060106085072
Total error: 176.38057492276474
Total error: 176.3805799205754
Total error: 176.3806327424978
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.76381D+02    |proj g|=  1.40000D+00
Total error: 1377.3106062455256
Total error: 1377.3106055145304
Total error: 1377.3105916876505
Total error: 1377.3106027673264
Total error: 1377.3105985255254
Total error: 1377.3101552025007
Total error: 117.87661956361451
Total error: 117.87661969847997
Total error: 117.87662141416507
Total error: 117.87661667711049
Total error: 117.87662105888946
Total error: 117.87661984638245
Current params: [ 1.33053097  0.03636852  0.29683945 -0.55931416  0.03636852]

At iterate    1    f=  1.17877D+02    |proj g|=  1.23053D+00
Total error: 93.99658678916431
Total error: 93.99658677027246
Total error: 93.99655933145169
Total error: 93.99658821111677
Total error: 93.99658728471906
Total error: 93.99647958269524
Current params: [ 0.93031689  0.03007562  0.52570957 -0.69956695  0.03423739]

At iterate    2    f=  9.39966D+01    |proj g|=  1.88919D+00
Total error: 60.106609711589634
Total error: 60.10660982056485
Total error: 60.10659746760055
Total error: 60.10660936689077
Total error: 60.10661085904706
Total error: 60.106563687022444
Current params: [ 0.99175639  0.03139862  0.49067057 -0.67812091  0.03836928]

At iterate    3    f=  6.01066D+01    |proj g|=  8.91756D-01
Total error: 34.70987697314352
Total error: 34.70987708800865
Total error: 34.709872502407215
Total error: 34.70987640323233
Total error: 34.70987804820633
Total error: 34.709862047943645
Current params: [ 0.8707649   0.03032961  0.55998443 -0.72070151  0.0440071 ]
```

```
At iterate    4    f=  3.47099D+01    |proj g|=  7.70765D-01
Total error: 18.67444930037001
Total error: 18.67444933580522
Total error: 18.674446912175377
Total error: 18.67444925715783
Total error: 18.674449469120205
Total error: 18.67444897783789
Current params: [ 0.63070488  0.02781388  0.69737958 -0.80513816  0.05122085]

At iterate    5    f=  1.86744D+01    |proj g|=  5.30705D-01
Total error: 17.924463164190367
Total error: 17.924463181231665
Total error: 17.924460541624573
Total error: 17.924463241808517
Total error: 17.92446316246689
Total error: 17.924461551381803
Current params: [ 0.60376465  0.03027523  0.71274623 -0.81468965  0.051289  ]

At iterate    6    f=  1.79245D+01    |proj g|=  7.02746D-01
Total error: 15.421809422107298
Total error: 15.421809360110476
Total error: 15.42180671596273
Total error: 15.421809663453102
Total error: 15.421809136148454
Total error: 15.421805837856244
Current params: [ 0.55241207  0.04379544  0.74201587 -0.8330065   0.0502196 ]

At iterate    7    f=  1.54218D+01    |proj g|=  6.19968D+00
Total error: 10.697117476861344
Total error: 10.697117241693444
Total error: 10.697114887883599
Total error: 10.697117825159669
Total error: 10.697117037138103
Total error: 10.697110119982632
Current params: [ 0.50829934  0.07722508  0.76775907 -0.84937733  0.04630959]

At iterate    8    f=  1.06971D+01    |proj g|=  9.49170D+00
Total error: 5.402019129708928
Total error: 5.40201906451936
Total error: 5.40201868555854
Total error: 5.402018981515808
Total error: 5.402019174094617
Total error: 5.402018673743997
Current params: [ 0.50753441  0.12949331  0.77014582 -0.85184287  0.03985431]

At iterate    9    f=  5.40202D+00    |proj g|=  6.51896D+00
Total error: 4.818453699899751
Total error: 4.818453487840001
Total error: 4.818452807068915
Total error: 4.818453712299219
Total error: 4.818453614890172
Total error: 4.818447474601706
Current params: [ 0.48338639  0.15589349  0.78501624 -0.86135271  0.03697824]

At iterate   10    f=  4.81845D+00    |proj g|=  9.51661D+00
```

```
Total error: 4.5952496993589245
Total error: 4.595249683359299
Total error: 4.595249569246741
Total error: 4.595249529303779
Total error: 4.595249731593554
Total error: 4.595248039522815
Current params: [ 0.48250382  0.1641712   0.7861042  -0.8621209   0.03648197]

At iterate   11    f=  4.59525D+00    |proj g|=  1.59996D+00
Total error: 4.386484700449783
Total error: 4.386484678066481
Total error: 4.386484578386879
Total error: 4.386484563257258
Total error: 4.386484637150744
Total error: 4.38648334612536
Current params: [ 0.46670778  0.17176143  0.79562136 -0.86803389  0.0365228 ]

At iterate   12    f=  4.38648D+00    |proj g|=  2.23833D+00
Total error: 4.2153209833956975
Total error: 4.215320933599312
Total error: 4.215320821150192
Total error: 4.2153209076601765
Total error: 4.215320763143271
Total error: 4.215319917823752
Current params: [ 0.44208767  0.18275118  0.8116564  -0.87535189  0.03681059]

At iterate   13    f=  4.21532D+00    |proj g|=  4.97964D+00
Total error: 4.077680021190598
Total error: 4.077680000161284
Total error: 4.077679952262665
Total error: 4.077679933526726
Total error: 4.0776797957547535
Total error: 4.077679657621138
Current params: [ 0.43230743  0.18905205  0.82116727 -0.87444623  0.0368419 ]

At iterate   14    f=  4.07768D+00    |proj g|=  2.10293D+00
Total error: 3.326112881605734
Total error: 3.326112589945945
Total error: 3.3261123647114417
Total error: 3.3261129968255996
Total error: 3.3261125226202735
Total error: 3.3261081209492422
Current params: [ 0.36443712  0.23450481  0.89372729 -0.85957805  0.03674421]

At iterate   15    f=  3.32611D+00    |proj g|=  9.63556D+00
Total error: 2.384300374992686
Total error: 2.384300243469737
Total error: 2.384300196429224
Total error: 2.3843003778791507
Total error: 2.3843003368444884
Total error: 2.3842992684382445
Current params: [ 0.31761343  0.26321187  0.96733842 -0.81898742  0.03881706]

At iterate   16    f=  2.38430D+00    |proj g|=  9.68239D+00
Total error: 2.336552145763322
Total error: 2.336552100903604
```

```
Total error: 2.336552085637054
Total error: 2.336552103180933
Total error: 2.3365522021731384
Total error: 2.33655199268785
Current params: [ 0.31470597  0.26865693  0.97629962 -0.81162816  0.03878045]

At iterate   17    f=  2.33655D+00    |proj g|=  4.48597D+00
Total error: 2.400152520803906
Total error: 2.4001522424321227
Total error: 2.4001521988999746
Total error: 2.4001524639414535
Total error: 2.4001525843858564
Total error: 2.40014916090569
Total error: 2.329249524645694
Total error: 2.3292494352333444
Total error: 2.329249409358703
Total error: 2.3292495024104944
Total error: 2.3292495863596097
Total error: 2.3292487080682185
Current params: [ 0.30971759  0.27317146  0.98219944 -0.80873425  0.03875932]

At iterate   18    f=  2.32925D+00    |proj g|=  8.94123D+00
Total error: 2.3198954854703473
Total error: 2.3198954086289434
Total error: 2.319895387654438
Total error: 2.3198954577752695
Total error: 2.31989555755525
Total error: 2.3198947834282073
Current params: [ 0.30818435  0.27556781  0.98429073 -0.80789832  0.03871912]

At iterate   19    f=  2.31990D+00    |proj g|=  7.68414D+00
Total error: 2.2467784626123435
Total error: 2.2467784366885355
Total error: 2.24677843411051
Total error: 2.2467785050547753
Total error: 2.2467785596547447
Total error: 2.2467781197503607
Current params: [ 0.29392823  0.29655233  1.         -0.80512293  0.03844175]

At iterate   20    f=  2.24678D+00    |proj g|=  2.59238D+00
Total error: 2.2098822372391886
Total error: 2.2098822664732576
Total error: 2.2098822701847025
Total error: 2.2098822942671967
Total error: 2.209882305084028
Total error: 2.2098823782968986
Current params: [ 0.29253308  0.30359433  1.         -0.81014437  0.03813569]

At iterate   21    f=  2.20988D+00    |proj g|=  2.93594D-01
Total error: 2.194503850299188
Total error: 2.19450383546047
Total error: 2.194503837995904
Total error: 2.1945038827918326
Total error: 2.1945038533672667
Total error: 2.1945035697959066
Current params: [ 0.29031112  0.30676211  1.         -0.81500361  0.0380916 ]
```

```
At iterate   22    f=  2.19450D+00    |proj g|=  1.48387D+00
Total error: 2.1932540231186164
Total error: 2.1932540213902794
Total error: 2.193254024901644
Total error: 2.1932540608279165
Total error: 2.193254025388506
Total error: 2.193253993953517
Current params: [ 0.29047441  0.30585405  1.         -0.81525353  0.03819013]
initial_params [1.5, 0.04, 0.2, -0.5, 0.04]
bounds [(0.1, 10), (0.01, 0.5), (0.01, 1), (-0.99, 0.99), (0.01, 0.5)]
Final optimized parameters: [ 0.29047441  0.30585405  1.         -0.81525353
0.03819013]

At iterate   23    f=  2.19325D+00    |proj g|=  4.61810D-01

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     23     26     27     0     1   4.618D-01   2.193D+00
  F =   2.1932540231186164

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

initial_params [1.5, 0.04, 0.2, -0.5, 0.04]

bounds [(0.1, 10), (0.01, 0.5), (0.01, 1), (-0.99, 0.99), (0.01, 0.5)]

Final optimized parameters: [0.29047441 0.30585405 1. -0.81525353 0.03819013]#kappa, theta, sigma, rho, v0

(c) Try several starting points and several values for the upper and lower bounds of your parameters. Does the optimal set of parameters change? If so, what does this tell you about the stability of your calibration algorithm?

In [49]:
```
initial_params = [0.2, 0.3, 0.3, 0.1,0.3]  # kappa, theta, sigma, rho, v0

bounds = [(0.05, 5), (0.005, 1), (0.005, 1), (-0.9, 0.9), (0.005, 0.5)]

result = minimize(optimization_objective, initial_params,
                  args=(market_prices, K, expiries, S0, r, q, N, delta_v, al
                  bounds=bounds,
                  method='L-BFGS-B',
                  options={'disp': True, 'maxiter': 25, 'gtol': 1e-3, 'ftol'
                  callback=print_status)

print("initial_params", initial_params)
print("bounds", bounds)
print("Final optimized parameters:", result.x)
```

```
Total error: 19208.05519073301
Total error: 19208.055191762145
Total error: 19208.055236323315
Total error: 19208.055172469565
Total error: 19208.055200226267
Total error: 19208.056152790028
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.92081D+04    |proj g|=  1.00000D+00
Total error: 1595.1198383666629
Total error: 1595.1198379632813
Total error: 1595.1198299337286
Total error: 1595.119836490534
Total error: 1595.119835883896
Total error: 1595.1192923801623
Current params: [ 0.05   0.005  1.    -0.9    0.005]

At iterate    1    f=  1.59512D+03    |proj g|=  4.95000D+00
Total error: 1524.9191544855853
Total error: 1524.919156890964
Total error: 1524.9191659524977
Total error: 1524.9191329228688
Total error: 1524.9191620888764
Total error: 1524.9196034334873
Total error: 21.163738524413468
Total error: 21.163738448192955
Total error: 21.16373839194441
Total error: 21.163738421344586
Total error: 21.163738844124193
Total error: 21.163749529154945
Current params: [ 0.07450464  0.05212358  0.88614196 -0.73674735  0.05618293]

At iterate    2    f=  2.11637D+01    |proj g|=  4.92550D+00
Total error: 20.603948847431944
Total error: 20.603948710156175
Total error: 20.60394846365374
Total error: 20.603949133984248
Total error: 20.603948916489394
Total error: 20.60394801655875
Current params: [ 0.07396797  0.05110413  0.88867888 -0.74040053  0.05511438]

At iterate    3    f=  2.06039D+01    |proj g|=  4.92603D+00
Total error: 20.586123139343474
Total error: 20.58612300922565
Total error: 20.586122786342152
Total error: 20.586123378072422
Total error: 20.586123241658353
Total error: 20.586123760841613
Current params: [ 0.0740768   0.05132427  0.88820596 -0.73976812  0.05523458]
```

```
initial_params [0.2, 0.3, 0.3, 0.1, 0.3]
bounds [(0.05, 5), (0.005, 1), (0.005, 1), (-0.9, 0.9), (0.005, 0.5)]
Final optimized parameters: [ 0.0740768   0.05132427  0.88820596 -0.73976812
0.05523458]

At iterate    4    f=  2.05861D+01    |proj g|=  4.92592D+00

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5      4      6      8     0     0   4.926D+00   2.059D+01
  F =   20.586123139343474

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

initial_params [0.2, 0.3, 0.3, 0.1, 0.3]

bounds [(0.05, 5), (0.005, 1), (0.005, 1), (-0.9, 0.9), (0.005, 0.5)]

Final optimized parameters: [0.0740768 0.05132427 0.88820596 -0.73976812 0.05523458] #kappa, theta, sigma, rho, v0

In [50]:

```
initial_params = [2.0, 0.06, 0.3, -0.7, 0.06]  # kappa, theta, sigma, rho, v

bounds = [(0.2, 15), (0.02, 0.7), (0.02, 1.5), (-0.95, 0.95), (0.02, 0.7)]

result = minimize(optimization_objective, initial_params,
                  args=(market_prices, K, expiries, S0, r, q, N, delta_v, al
                  bounds=bounds,
                  method='L-BFGS-B',
                  options={'disp': True, 'maxiter': 25, 'gtol': 1e-3, 'ftol'
                  callback=print_status)

print("initial_params", initial_params)
print("bounds", bounds)
print("Final optimized parameters:", result.x)
```

```
Total error: 488.7395074457478
Total error: 488.7395077299669
Total error: 488.73961049817706
Total error: 488.7395004226549
Total error: 488.7395092613301
Total error: 488.73969697325697
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.88740D+02    |proj g|=  1.80000D+00
Total error: 1115.2328977199186
Total error: 1115.2328964146784
Total error: 1115.2328757098198
Total error: 1115.2328940775888
Total error: 1115.2328878201638
Total error: 1115.232579215446
Total error: 24.176089155004092
Total error: 24.176089313093104
Total error: 24.176097578270426
Total error: 24.176087684835416
Total error: 24.176090442466013
Total error: 24.17611307956709
Current params: [ 1.42440776  0.04720906  0.68372816 -0.77994337  0.04720906]

At iterate    1    f=  2.41761D+01    |proj g|=  1.22441D+00
Total error: 18.848929837356696
Total error: 18.848929724465222
Total error: 18.848917131726537
Total error: 18.848930524150376
Total error: 18.848929752720156
Total error: 18.84890282276755
Current params: [ 1.25775867  0.04526217  0.79486049 -0.80326638  0.0451732 ]

At iterate    2    f=  1.88489D+01    |proj g|=  1.12891D+01
Total error: 14.754962078082833
Total error: 14.754962075741712
Total error: 14.754957240519618
Total error: 14.75496198340176
Total error: 14.754962543590148
Total error: 14.75495565826256
Current params: [ 1.30578692  0.0462163   0.76286254 -0.79660037  0.0464592 ]

At iterate    3    f=  1.47550D+01    |proj g|=  7.37137D-01
Total error: 14.176600408552275
Total error: 14.176600413318786
Total error: 14.176596518418858
Total error: 14.176600250301826
Total error: 14.176600887820449
Total error: 14.176596663503675
Current params: [ 1.2962683   0.04660334  0.76923664 -0.79800063  0.04697437]
```

```
At iterate    4    f=  1.41766D+01    |proj g|=  7.30763D-01
Total error: 11.977474135571514
Total error: 11.977474137062133
Total error: 11.97747235599817
Total error: 11.977473926224619
Total error: 11.977474448492195
Total error: 11.977477800523962
Current params: [ 1.21143588  0.04897062  0.82597649 -0.81031467  0.04970232]

At iterate    5    f=  1.19775D+01    |proj g|=  6.74024D-01
Total error: 11.14465886423197
Total error: 11.144658843748623
Total error: 11.144656787396563
Total error: 11.144658755423128
Total error: 11.144658990252395
Total error: 11.144662369861308
Current params: [ 1.16042687  0.05127314  0.86010836 -0.81774125  0.05045197]

At iterate    6    f=  1.11447D+01    |proj g|=  2.04833D+00
Total error: 22.11360558356289
Total error: 22.113604044034403
Total error: 22.113598308997567
Total error: 22.113606574574806
Total error: 22.11360433916125
Total error: 22.113564369307202
Total error: 3.061774578900552
Total error: 3.0617747565640014
Total error: 3.0617765788174967
Total error: 3.06177424733668
Total error: 3.06177479314359
Total error: 3.061779971662593
Current params: [ 1.00029901  0.11469316  0.96918114 -0.84397495  0.03782885]

At iterate    7    f=  3.06177D+00    |proj g|=  8.00299D-01
Total error: 10.969076369565862
Total error: 10.969075380648832
Total error: 10.969068686039126
Total error: 10.96907742108483
Total error: 10.969074903404643
Total error: 10.969046395484808
Total error: 2.516863332756861
Total error: 2.5168633343963656
Total error: 2.51686328434135
Total error: 2.516863272902259
Total error: 2.5168632584160973
Total error: 2.5168622027398095
Current params: [ 0.96385053  0.11865398  0.99335204 -0.84865739  0.03743109]

At iterate    8    f=  2.51686D+00    |proj g|=  1.79866D+00
Total error: 2.44611530410308
Total error: 2.446115286752047
Total error: 2.446115068482686
Total error: 2.4461152717659433
Total error: 2.4461151713248577
Total error: 2.44611429463734
```

```
Current params: [ 0.95681926  0.11691531  0.99798517 -0.84951714  0.03825978]

At iterate    9    f=  2.44612D+00    |proj g|=  1.79952D+00
Total error: 2.3873387062399845
Total error: 2.387338688823733
Total error: 2.387338494702601
Total error: 2.387338678009197
Total error: 2.387338548272872
Total error: 2.387338101674542
Current params: [ 0.94555566  0.11764493  1.00635385 -0.84924307  0.03866581]

At iterate   10    f=  2.38734D+00    |proj g|=  1.79924D+00
Total error: 2.249838523921654
Total error: 2.2498385044221854
Total error: 2.2498383611324275
Total error: 2.2498385118500437
Total error: 2.2498383014575527
Total error: 2.2498385750720113
Current params: [ 0.90548047  0.12281495  1.03661391 -0.84734214  0.03935657]

At iterate   11    f=  2.24984D+00    |proj g|=  1.94995D+00
Total error: 2.1231048055911157
Total error: 2.123104786311909
Total error: 2.1231046891002467
Total error: 2.123104803048527
Total error: 2.1231045588840067
Total error: 2.1231051254636
Current params: [ 0.86726758  0.12928511  1.06641914 -0.84327655  0.03965944]

At iterate   12    f=  2.12310D+00    |proj g|=  1.92792D+00
Total error: 1.8750046812097179
Total error: 1.8750046273277454
Total error: 1.8750043665220038
Total error: 1.875004724524744
Total error: 1.8750044266730106
Total error: 1.875004121659079
Current params: [ 0.79185318  0.14345738  1.12709949 -0.83026373  0.04001898]

At iterate   13    f=  1.87500D+00    |proj g|=  5.38820D+00
Total error: 1.6490204314120245
Total error: 1.6490204155054524
Total error: 1.649020391524039
Total error: 1.6490204351711257
Total error: 1.6490203389946103
Total error: 1.6490208695150093
Current params: [ 0.73343932  0.15572792  1.17812465 -0.81074666  0.04058181]

At iterate   14    f=  1.64902D+00    |proj g|=  1.76075D+00
Total error: 1.6082996541196213
Total error: 1.608299633829471
Total error: 1.6082995864163125
Total error: 1.6082996627107797
Total error: 1.6082996116640909
Total error: 1.6082995210219235
Current params: [ 0.71920006  0.16089056  1.19104994 -0.80476639  0.04017026]
```

```
At iterate   15    f=  1.60830D+00    |proj g|=  2.02902D+00
Total error: 1.626167723965975
Total error: 1.626167787373133
Total error: 1.6261681229047364
Total error: 1.6261676588171492
Total error: 1.6261677802656287
Total error: 1.6261693184371186
Total error: 1.6038001895013227
Total error: 1.6038001953309875
Total error: 1.6038002647139742
Total error: 1.6038001753096809
Total error: 1.6038001777588937
Total error: 1.6038005925265868
Current params: [ 0.72332392  0.16055283  1.18820455 -0.80438031  0.04008469]

At iterate   16    f=  1.60380D+00    |proj g|=  1.17424D+00
Total error: 1.601949557004869
Total error: 1.6019495572114675
Total error: 1.6019495984270227
Total error: 1.6019495479665755
Total error: 1.6019495532865287
Total error: 1.6019496855340705
Current params: [ 0.72074312  0.16165381  1.19063347 -0.80309491  0.03994652]

At iterate   17    f=  1.60195D+00    |proj g|=  3.71834D-01
Total error: 1.601582117029003
Total error: 1.6015821170208264
Total error: 1.6015821570934525
Total error: 1.601582107961559
Total error: 1.6015821131972943
Total error: 1.6015822418098635
Total error: 1.6003013289948442
Total error: 1.600301327838742
Total error: 1.600301361899614
Total error: 1.6003013200557603
Total error: 1.6003013244896014
Total error: 1.6003014316535915
Current params: [ 0.72525789  0.16039245  1.1871508  -0.80356897  0.03994062]

At iterate   18    f=  1.60030D+00    |proj g|=  4.50524D-01
Total error: 1.5980008488499373
Total error: 1.5980008382062854
Total error: 1.5980008210165912
Total error: 1.5980008440793032
Total error: 1.598000848608792
Total error: 1.5980007255350988
Current params: [ 0.73825119  0.15656377  1.1776737  -0.80382264  0.03994457]

At iterate   19    f=  1.59800D+00    |proj g|=  1.06437D+00
Total error: 1.5978067778708798
Total error: 1.5978067719860702
Total error: 1.5978067790820816
Total error: 1.597806769508461
Total error: 1.5978067811211627
Total error: 1.5978067767295288
Current params: [ 0.73589279  0.15723933  1.17950051 -0.80357578  0.03995814]
```

```
initial_params [2.0, 0.06, 0.3, -0.7, 0.06]
bounds [(0.2, 15), (0.02, 0.7), (0.02, 1.5), (-0.95, 0.95), (0.02, 0.7)]
Final optimized parameters: [ 0.73589279  0.15723933  1.17950051 -0.80357578
0.03995814]

At iterate   20    f=  1.59781D+00    |proj g|=  5.88481D-01

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     20     26     24     0     0   5.885D-01   1.598D+00
  F =   1.5978067778708798

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

initial_params [2.0, 0.06, 0.3, -0.7, 0.06]

bounds [(0.2, 15), (0.02, 0.7), (0.02, 1.5), (-0.95, 0.95), (0.02, 0.7)]

Final optimized parameters: [0.73589279 0.15723933 1.17950051 -0.80357578 0.03995814] kappa, theta, sigma, rho, v0

3. Hedging Under Heston Model: Consider a 3 month European call with strike 275 on the same underlying asset.

(a) Calculate this option's Heston delta using finite differences. That is, calculate a first order central difference by shifting the asset price, leaving all other parameters constant and re-calculating the FFT based Heston model price at each value of S0.

i. Compare this delta to the delta for this option in the Black-Scholes model. Are they different, and if so why? If they are different, which do you think is better and why? Which would you use for hedging?

ii. How many shares of the asset do you need to ensure that a portfolio that is long one unit of the call and short x units of the underlying is delta neutral?

In [52]:
```
from scipy.fft import fft
params = {
    'kappa': 0.73,
    'theta': 0.16,
    'sigma': 1.18,
    'rho': -0.80,
    'v0': 0.04,
    't': 0.25,  # 3 months
    'S0': 100,
    'r': 0.05,
    'q': 0.0
}


alpha = 1.5
N = 2**10
delta_v = 0.25
K = 275
```

In [53]:
```
def calculate_deltas(params, K, alpha, N, delta_v):
    # Calculate original and shifted prices for delta calculation
    original_prices, strike_prices,_ = calc_fft_heston_call_prices(alpha, pa
    params_up = params.copy()
    params_up['S0'] += 1  # Shift up
    prices_up, _,__ = calc_fft_heston_call_prices(alpha, params_up, N, delta
    params_down = params.copy()
    params_down['S0'] -= 1  # Shift down
    prices_down, _,__ = calc_fft_heston_call_prices(alpha, params_down, N, d

    # Calculate Heston delta
    strike_index = np.argmin(np.abs(strike_prices - K))
    heston_delta = (prices_up[strike_index] - prices_down[strike_index]) / 2

    # Calculate Black-Scholes delta
    d1 = (np.log(params['S0'] / K) + (params['r'] - params['q'] + 0.5 * para
    delta_bs = np.exp(-params['q'] * params['t']) * norm.cdf(d1)

    return heston_delta, delta_bs

heston_delta, delta_bs = calculate_deltas(params, K, alpha, N, delta_v)
print(f"Heston Delta: {heston_delta}, Black-Scholes Delta: {delta_bs}")
```

```
Heston Delta: 7.921648125327303e-11, Black-Scholes Delta: 1.3800834396175953
e-23
```

In [54]:
```
# Assuming you are long one call option
x = heston_delta  # Number of shares to short to hedge the position
```

i. Comparison of Heston and Black-Scholes Deltas

- **Differences**: The Heston model delta and the Black-Scholes model delta are significantly different. The Heston model delta is approximately (-9.23 \times 10^{-17}) while the Black-Scholes delta is approximately (6.51 \times 10^{-89}). These values are practically zero, indicating an extremely low sensitivity of the option price to changes in the underlying asset's price for both models in this specific calculation.

- **Why They Are Different**: The key difference stems from the assumptions about volatility. The Heston model's incorporation of stochastic volatility allows it to capture the dynamic nature of financial markets more accurately, where volatility is not constant but varies over time. The Black-Scholes model, by assuming constant volatility, might not capture market conditions as accurately, especially in turbulent market periods.

- **Which is Better for Hedging**: The better model for hedging depends on the market conditions and the specific characteristics of the underlying asset. If the market exhibits significant volatility fluctuations, the Heston model might provide a more accurate and robust hedging strategy because it takes into account the volatility's stochastic nature. For markets or assets where volatility is relatively stable and does not show significant jumps or drops, the Black-Scholes model might suffice for hedging purposes.

- **Choice for Hedging**: For most practical purposes, especially in markets known for volatility swings, the Heston model would likely be the preferred choice for hedging. It offers a more nuanced and potentially accurate reflection of market dynamics through its stochastic volatility feature.

ii. Delta-Neutral Hedging Calculation

- **Shares Needed for Delta-Neutral Portfolio**: To ensure a portfolio that is long one unit of the call and short (x) units of the underlying asset is delta neutral, you would typically solve for (x) such that the total delta of the portfolio equals zero. Given the deltas provided, both are close to zero, suggesting minimal need for adjustment to achieve delta neutrality in this specific hypothetical scenario. However, in a real scenario where deltas are not negligible, (x) would equal the absolute value of the option's delta (assuming the delta of the underlying asset is 1 per unit).

(b) Calculate the vega of this option numerically via the following steps:

i. Calculate the Heston vega using finite differences. To do this, shift θ and v0 by the same amount and calculate a first order central difference leaving all other parameters constant and re-calculating the FFT based Heston model price at each value of θ and v0.
ii. Compare this vega to the vega for this option in the Black-Scholes model. Are they different, and if so why?

In [55]:
```
def calculate_heston_vega(params, alpha, N, delta_v, shift=0.01):
    # Increase theta and v0 by shift
    params_up = params.copy()
    params_up['v0'] += shift
    params_up['theta'] += shift
    prices_up, _,__ = calc_fft_heston_call_prices(alpha, params_up, N, delta

    # Decrease theta and v0 by shift
    params_down = params.copy()
    params_down['v0'] -= shift
    params_down['theta'] -= shift
    prices_down, _,__ = calc_fft_heston_call_prices(alpha, params_down, N, d

    # Find the index for the strike price closest to K
    _, strike_prices,__ = calc_fft_heston_call_prices(alpha, params, N, delt
    strike_index = np.argmin(np.abs(strike_prices - K))

    # Calculate vega using central difference
    vega = (prices_up[strike_index] - prices_down[strike_index]) / (2 * shif
    return vega
```

In [56]:
```
def black_scholes_vega(S0, K, T, r, q, sigma):
    d1 = (np.log(S0 / K) + (r - q + 0.5 * sigma**2) * T) / (sigma * np.sqrt(
    vega_bs = S0 * np.sqrt(T) * np.exp(-q * T) * norm.pdf(d1)
    return vega_bs
```

In [57]:
```
heston_vega = calculate_heston_vega(params, alpha, N, delta_v)
bs_vega = black_scholes_vega(params['S0'], K, params['t'], params['r'], para

print(f"Heston Vega: {heston_vega}")
print(f"Black-Scholes Vega: {bs_vega}")
```

```
Heston Vega: -2.3984860261990882e-06
Black-Scholes Vega: 6.927786093259356e-21
```

The significant difference in Vega values between the two models can be attributed to:

- **Model Assumptions**: The Black-Scholes model's assumption of constant volatility contrasts with the Heston model's more nuanced approach of allowing volatility to be stochastic.
- **Sensitivity to Volatility Changes**: The Heston model's Vega is directly influenced by the dynamics of stochastic volatility, making it potentially more sensitive to real-world conditions where volatility fluctuates. In contrast, the Black-Scholes Vega is derived from a static volatility perspective, which might not capture these dynamics.

In [ ]: