M86 Homework 3

Alex Craig & Ganqi Li

Exercise 1

Wilmott, Chapter 8, Problems 1, 2, 9, 14

Question 8.1 Find the explicit value for the value of a European option with payoff $\Lambda(S)$ and expiry time T, where:

$$\Lambda(S) = \begin{cases} S & \text{if } S > E \\ 0 & \text{if } S < E \end{cases}$$

Solution The payoff function can be rewritten as:

$$\Lambda(S) = \max(E - S, 0) - E\mathcal{H}(E - S) + S$$

The option is therefore the combination of a long put, a short binary put, and a long share. Both options have strike E and expire at time T. We can therefore use the explicit values of each of these options to find the value of the European option.

The value of the long put is given by:

$$V_{\text{put}} = -SN(-d_1) + Ee^{-r(T-t)}N(-d_2)$$

The value of the short binary put is given by:

$$V_{\text{binary}} = -Ee^{-r(T-t)}\left(1 - N(-d_2)\right)$$

The value of the long share is given by:

$$V_{\text{share}} = S$$

The value of the European option is therefore:

$$V(S,t) = V_{\text{put}} - V_{\text{binary}} + V_{\text{share}}$$

Let us simplify the expression:

$$V(S,t) = -SN(-d_1) + Ee^{-r(T-t)}N(-d_2) - Ee^{-r(T-t)}\left(1 - N(-d_2)\right) + S$$

$$V(S,t) = -SN(-d_1) + Ee^{-r(T-t)}N(-d_2) - Ee^{-r(T-t)}N(-d_2) + S$$

$$V(S,t) = S\left(1 - N(-d_1)\right) = SN(d_1)$$

where:

$$d_1 = \frac{\log(\frac{S}{E}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{(T-t)}}$$

Question 8.2 Find the explicit solution for the value of a European supershare option, with expiry at time T and payoff:

$$\Lambda(S) = \mathcal{H}(S - E_1) - \mathcal{H}(S - E_2)$$

where $E_1 < E_2$

Solution In this question, we are essentially long a binary call option with strike E_1 and short a binary call option with strike E_2. The price of a binary call option is given by:

$$V(S,t) = e^{-r(T-t)}N(d_2)$$

where:

$$d_2(E) = \frac{\ln(\frac{S}{E}) + (r - \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}$$

Therefore, the value of the supershare option is given by the value of the long binary call option minus the value of the short binary call option:

$$C(S,t) = e^{-r(T-t)}\left(N(d_2(E_1)) - N(d_2(E_2))\right)$$

Question 8.9 A forward start call option is specified as follows: at time T_1, the holder is given a European call option with exercise price $S(T_1)$ and expiry at time $T_1 + T_2$. What is the value of the option for $0 \leq t \leq T_1$?

Solution At time T_1, the contract becomes a vanilla European call option with strike $E = S(T_1)$ and current stock price $S(T_1)$ so we know that the value at time T_1 is given by:

$$V(S(T_1), T_1) = S(T_1)\left(N(d_1) - e^{-r(T_2-T_1)}N(d_2)\right)$$

where:

$$d_1 = \frac{\ln(\frac{S(T_1)}{S(T_1)}) + (r + \frac{\sigma^2}{2})(T_2 - T_1)}{\sigma\sqrt{T_2 - T_1}} = \frac{(r + \frac{\sigma^2}{2})(T_2 - T_1)}{\sigma\sqrt{T_2 - T_1}}$$

$$d_2 = d_1 - \sigma\sqrt{T_2 - T_1}$$

Therefore, the option at time T_1 is equal to a constant multiplied by the asset price. For no-arbitrage to hold, the value of the option at time t must be the same.

$$V(S,t) = S(t)\left(N(d_1) - e^{-r(T_2-t)}N(d_2)\right)$$

Question 8.14 Use put-call parity to find the relationships between deltas, gammas, vegas, thetas, and rhos for European call and put options.

Solution The put-call parity formula for European options is:

$$C - P = S - Ee^{-r(T-t)}$$

where:

- C is the price of the European call option,
- P is the price of the European put option,
- S is the current price of the underlying asset,
- r is the risk-free interest rate,
- T is the time to expiration,
- E is the strike price.

From this, we can derive the relationships between the Greeks for European call and put options.

Delta (Δ)

Delta measures the sensitivity of the option's price to a change in the price of the underlying asset:

$$\frac{\partial(C-P)}{\partial S} = 1 \implies \Delta_C = 1 + \Delta_P$$

Gamma (Γ)

Gamma measures the rate of change of delta with respect to changes in the underlying price:

$$\frac{\partial^2(C-P)}{\partial S^2} = 0 \implies \Gamma_C = \Gamma_P$$

Vega ($\mathcal{V}$)

Vega measures sensitivity to volatility:

$$\frac{\partial(C-P)}{\partial \sigma} = 0 \implies \mathcal{V}_C = \mathcal{V}_P$$

Theta (Θ)

Theta measures sensitivity to the passage of time:

$$\frac{\partial(C-P)}{\partial t} = -rEe^{-r(T-t)} \implies \Theta_C = \Theta_P - rEe^{-r(T-t)}$$

Rho (ρ)

Rho measures sensitivity to the interest rate:

$$\frac{\partial(C-P)}{\partial r} = E(T-t)e^{-r(T-t)} \implies \rho_C = \rho_P + E(T-t)e^{-r(T-t)}$$

Exercise 2

The Breeden-Litzenberger formula for the pdf $\phi(x)$ of the future price S_T under the risk-neutral measure Q is given by

$$\phi(x) = \frac{\partial^2 C}{\partial K^2}$$

where $C = C(K,T)$ is the market price of a call option as a function of strike K and expiration T.

(a) For a stock of your choice, choose the nearest available expiration date that is at least 4 weeks in the future and download the call prices for all strikes at that expiration. Use the package `scipy.interpolate.CubicSpline` to interpolate the list of call prices using a "natural" cubic spline (use `bc_type`). Use the mid prices (average of bid and ask). Plot the prices and the interpolating spline in a single graph.

```
## repeated printouts
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import pandas as pd

# df_calls = pd.read_excel("TSLA_calls_220901.xlsx")
df_calls = pd.read_excel("AAPL_calls_220901.xlsx")

## Subset df_calls to the nearest vailable expiration date that's at least 4 weeks into the future
df_calls['days_to_expire'] = (df_calls['expiration_date'] - df_calls['current_date']).dt.days

# df_calls = df_calls[df_calls['days_to_expire'] > 30]
# df_calls = df_calls[df_calls['days_to_expire'] < 50]
```

```
df_calls = df_calls[df_calls['days_to_expire'] > 28]
df_calls = df_calls[df_calls['days_to_expire'] < 30]


## Create bid-ask mean values and strike price
df_calls['mid_price'] = (df_calls['highest_bid'] + df_calls['lowest_ask'])/2
df_calls['strike'] = df_calls['strike_times_1000']/1000

df_calls.head(1)
```

```
    current_date expiration_date type  strike_times_1000  highest_bid
257   2022-09-01      2022-09-30    C             100000        58.15  \

     lowest_ask  implied_vol Ticker Symbol  days_to_expire  mid_price  strike
257        58.4     0.723393          AAPL              29     58.275   100.0
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

plt.figure(figsize=(8, 6))

## Sort the DataFrame by strike price
df_calls.sort_values(by='strike', inplace=True)

## Extract strike prices and mid prices
strike_prices = df_calls['strike']
mid_prices = df_calls['mid_price']

## Perform cubic spline interpolation
spline = CubicSpline(strike_prices, mid_prices, bc_type='natural')

min_strike = np.min(strike_prices)
max_strike = np.max(strike_prices)

## Generate finer strike prices for smooth plotting
fine_strike_prices = np.linspace(min_strike, max_strike, 100)  # Use np.min and np.max here

## Plot the original data points
plt.scatter(strike_prices, mid_prices, color='black', label='Original Prices')

## Plot the interpolating spline
plt.plot(fine_strike_prices, spline(fine_strike_prices), label='Interpolating Spline')

plt.title('Interpolating Call Option Prices on 22/09/01')
plt.xlabel('Strike Price')
plt.ylabel('Middle of Bid-Ask')
plt.legend()
plt.grid(True)
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

```
<matplotlib.collections.PathCollection at 0x2993b7550>
```

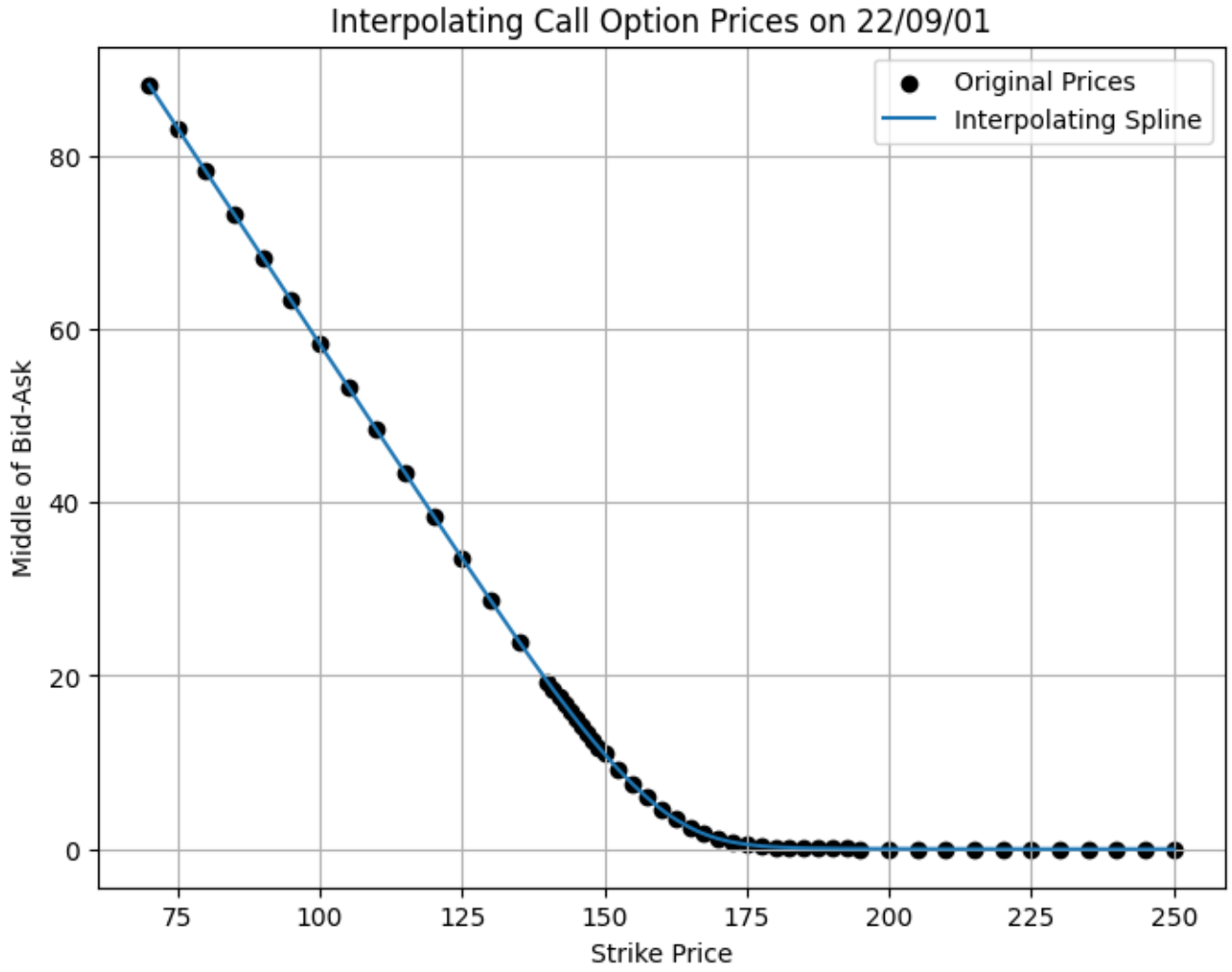
```
[<matplotlib.lines.Line2D at 0x2a46e0250>]
```

```
Text(0.5, 1.0, 'Interpolating Call Option Prices on 22/09/01')
```

```
Text(0.5, 0, 'Strike Price')
```

```
Text(0, 0.5, 'Middle of Bid-Ask')
```

```
<matplotlib.legend.Legend at 0x2993403d0>
```



(b) Use `scipy.interpolate.CubicSpline.derivative` to create a plot of the second derivative $\frac{\partial^2 C}{\partial K^2}$ of the cubic spline. This graph is a representation of the risk-neutral density for the stock price S_T that is implicit in the market prices of the call options.

Answer:

We plotted graphs for both the first derivative and the second derivative, and the first derivative graph shares similar characteristics with a CDF curve.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

plt.figure(figsize=(8, 6))

## Calculate the first derivative (slope) of the cubic spline
spline_first_derivative = spline.derivative()

## Evaluate the first derivative at each strike price
first_derivative_values = spline_first_derivative(strike_prices)

## Plot the second derivative against the strike prices
plt.plot(strike_prices, first_derivative_values)
plt.title(r'First Derivative, $\frac{dC}{dK}$')
plt.xlabel('Strike Price')
plt.ylabel('First Derivative')
```

```
plt.grid(True)
plt.show()
```

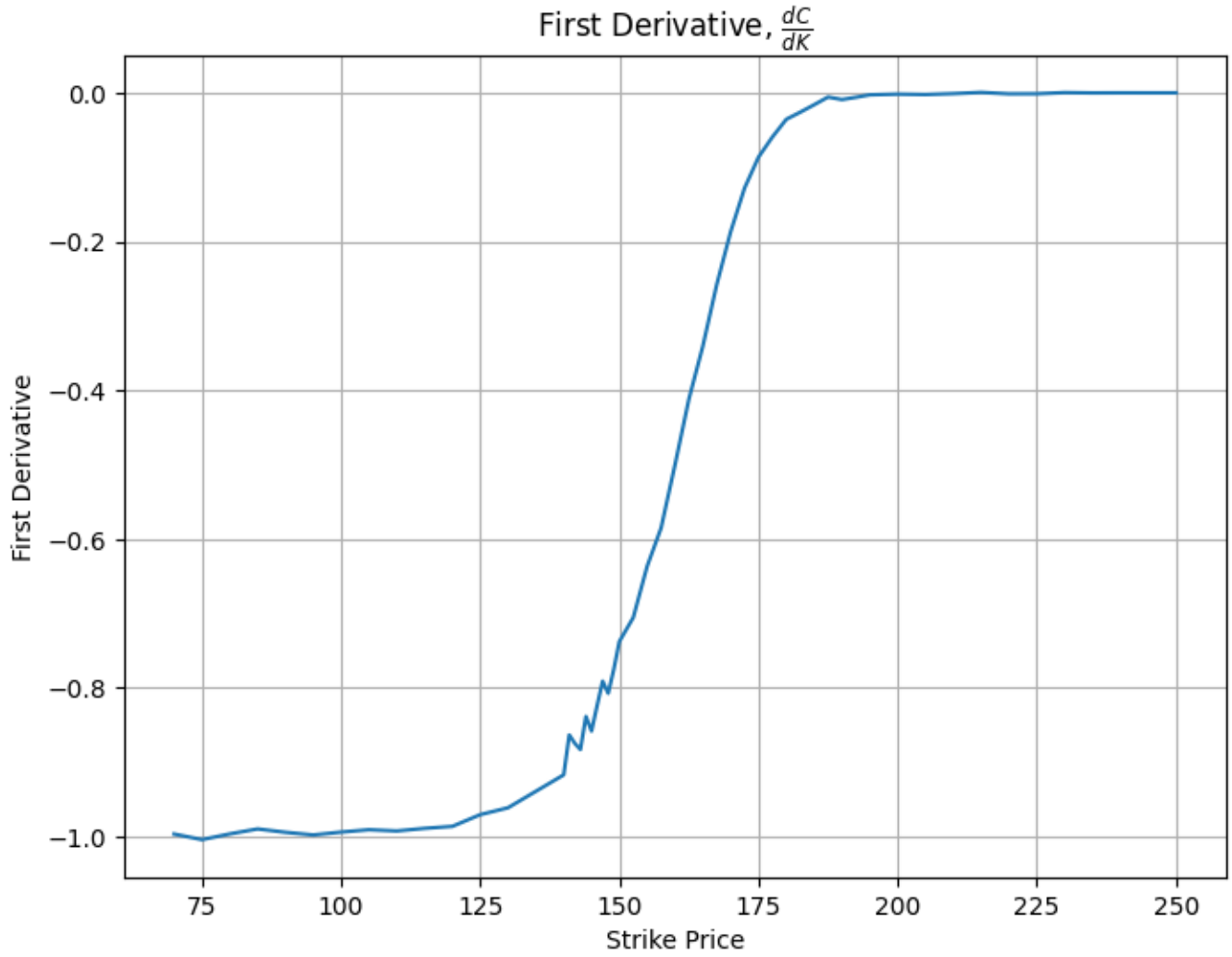
```
<Figure size 800x600 with 0 Axes>
```

```
[<matplotlib.lines.Line2D at 0x2993bb810>]
```

```
Text(0.5, 1.0, 'First Derivative, $\\frac{dC}{dK}$')
```

```
Text(0.5, 0, 'Strike Price')
```

```
Text(0, 0.5, 'First Derivative')
```




```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

plt.figure(figsize=(8, 6))

## Calculate the first derivative (slope) of the cubic spline
spline_first_derivative = spline.derivative()

## Calculate the second derivative of the cubic spline (curvature)
spline_second_derivative = spline_first_derivative.derivative()

## Evaluate the second derivative at each strike price
second_derivative_values = spline_second_derivative(strike_prices)

## Plot the second derivative against the strike prices
```

```
plt.plot(strike_prices, second_derivative_values)
plt.title(r'Second Derivative, $\frac{d^2C}{dK^2}$')
plt.xlabel('Strike Price')
plt.ylabel('Second Derivative (Density)')

plt.grid(True)
plt.show()
```

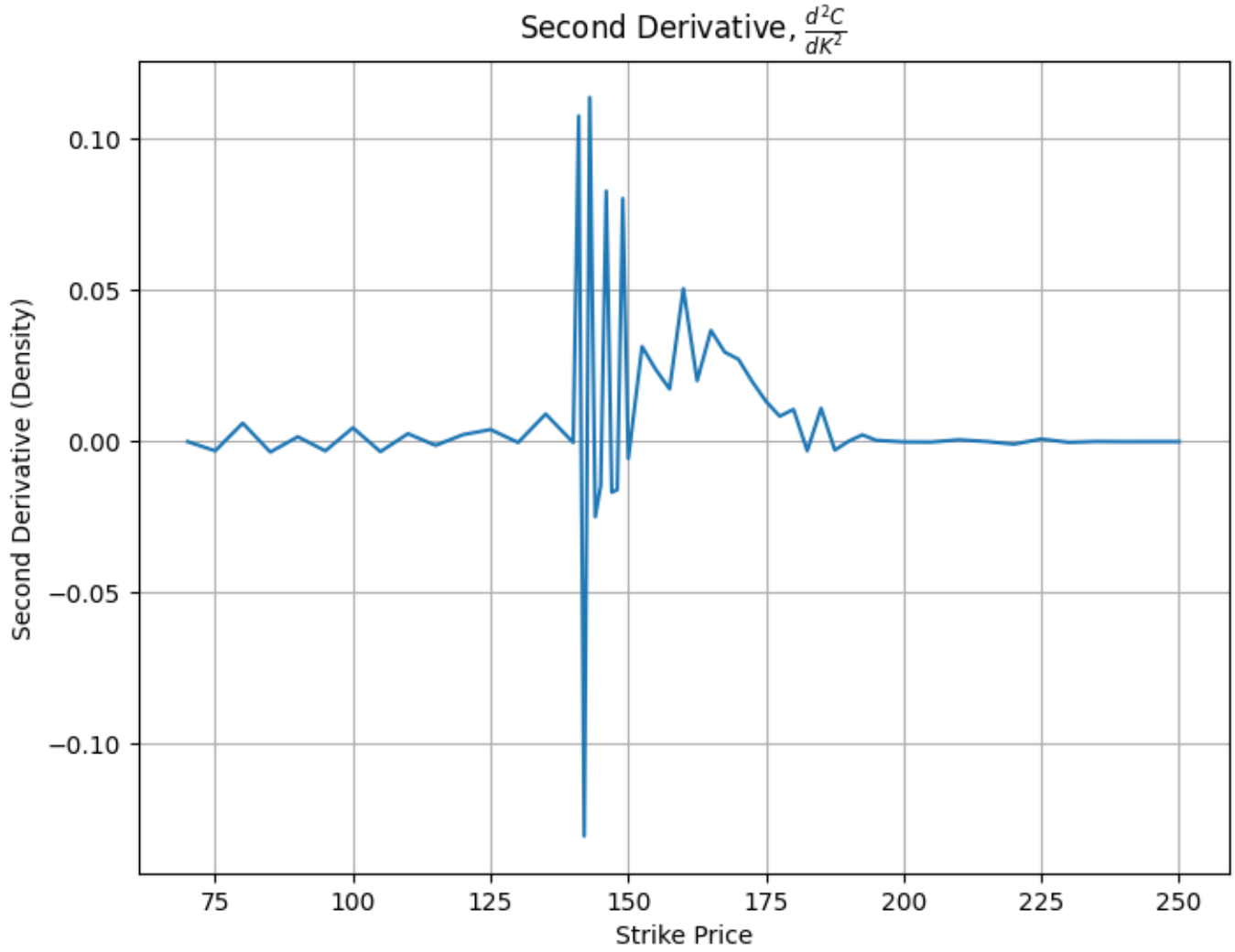
```
<Figure size 800x600 with 0 Axes>
```

```
[<matplotlib.lines.Line2D at 0x2aa396f10>]
```

```
Text(0.5, 1.0, 'Second Derivative, $\\frac{d^2C}{dK^2}$')
```

```
Text(0.5, 0, 'Strike Price')
```

```
Text(0, 0.5, 'Second Derivative (Density)')
```



(c) Create a second plot with a logarithmic x-axis. Does the distribution seem log-normal or is it skewed (and if so in which direction)?

Answer:

Since we only used options data within a single day for a single stock, the second derivative graph looks messy and produces negative density values, while the first derivative graph looks much cleaner in comparison. When plotted with a logarithmic x-axis, the second derivative (density) appears to skew to the left.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy.interpolate import CubicSpline

plt.figure(figsize=(8, 6))

## Calculate the first derivative (slope) of the cubic spline
spline_first_derivative = spline.derivative()

## Calculate the second derivative of the cubic spline (curvature)
spline_second_derivative = spline_first_derivative.derivative()

## Evaluate the second derivative at each strike price
second_derivative_values = spline_second_derivative(strike_prices)

## Plot the second derivative against the strike prices
plt.plot(strike_prices, second_derivative_values)
plt.title(r'Second Derivative, $\frac{d^2C}{dK^2}$')
plt.xlabel('Strike Price (Log-Scale)')
plt.ylabel('Second Derivative (Density)')


plt.xscale('log')

plt.grid(True)
plt.show()
```

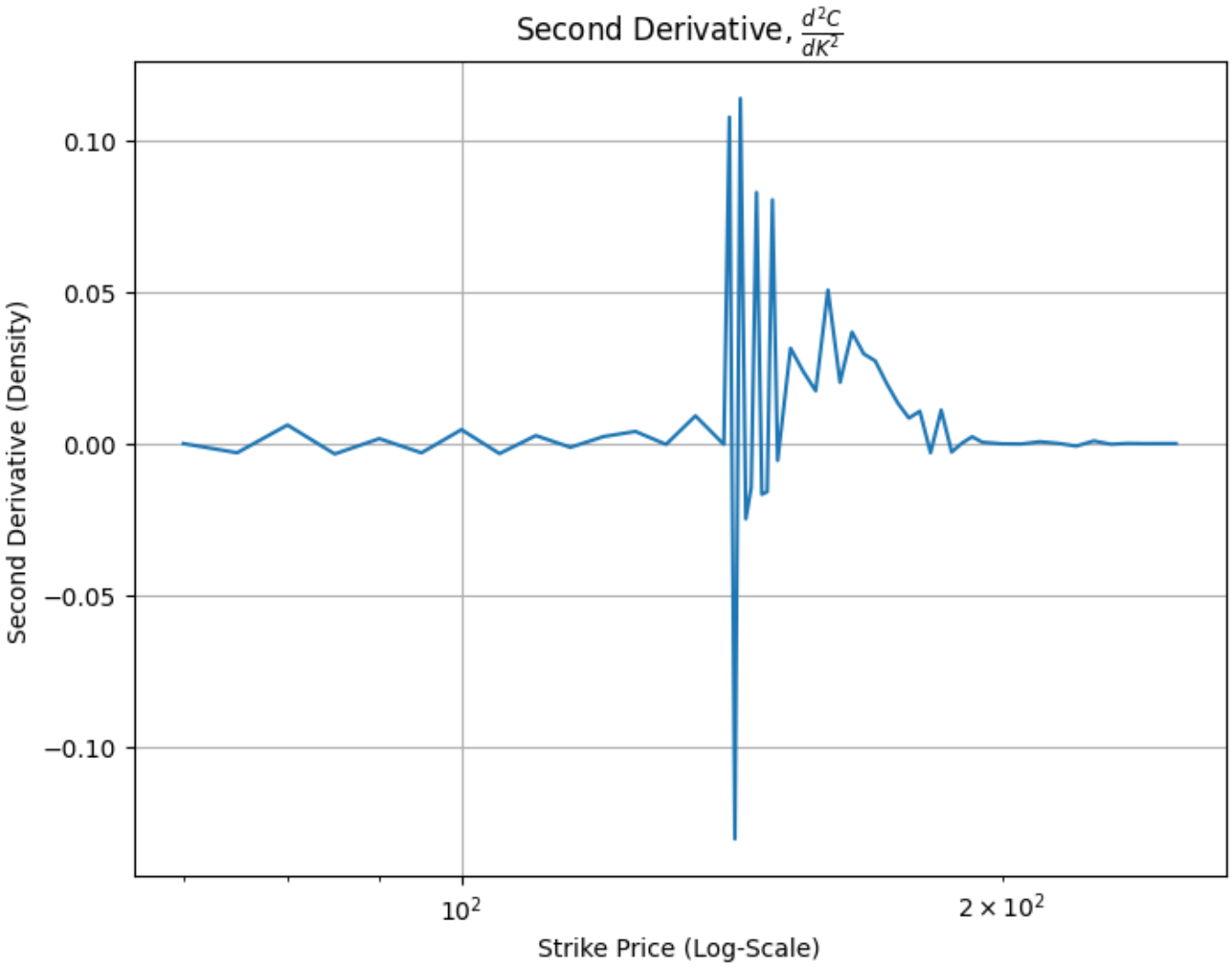
```
<Figure size 800x600 with 0 Axes>
```

```
[<matplotlib.lines.Line2D at 0x2996fca10>]
```

```
Text(0.5, 1.0, 'Second Derivative, $\\frac{d^2C}{dK^2}$')
```

```
Text(0.5, 0, 'Strike Price (Log-Scale)')
```

```
Text(0, 0.5, 'Second Derivative (Density)')
```



Exercise 3

(a) Implement a function that calculates implied volatility using Newton's method. For example, you can follow Wilmot's sample code. Plot the implied volatilities of the call options you used in the previous problem. As interest rate, use the rate of a US Treasury bond with expiration close to the expiration of your call options (e.g., 1 month). Note that Newton's method may or may not converge depending on your initial guess for the volatility. You may need to experiment to get your code to converge for all prices in your range.

```
## Set 1-month interest rate on 22/09/01 from FRED
interest_rate = 2.53/100

## Set AAPL and TSLA prices on 22/09/01 from CRSP
AAPL_price = 157.9600067
# TSLA_price = 277.1600037

import numpy as np
from scipy.stats import norm
from scipy.optimize import newton

## Define function for theoretical value
def black_scholes_call(S, K, T, r, sigma):
    """
    Calculates the theoretical price of a European call option using the Black-Scholes formula.

    Parameters:
    - S: Current price of the underlying asset (e.g., stock price).
```

```
    - K: Strike price of the option.
    - T: Time to expiration of the option in years.
    - r: Risk-free interest rate (annualized, expressed as a decimal).
    - sigma: Volatility of the underlying asset's returns (annualized, expressed as a decimal).

    Returns:
    - Theoretical price of the European call option calculated using the Black-Scholes formula.
    """
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return call_price


def implied_volatility(S, K, T, r, call_price):
    """
    Calculates the implied volatility of a European call option using Newton's method.

    Parameters:
    - S: Current price of the underlying asset (e.g., stock price).
    - K: Strike price of the option.
    - T: Time to expiration of the option in years.
    - r: Risk-free interest rate (annualized, expressed as a decimal).
    - call_price: Observed market price of the call option.

    Returns:
    - Implied volatility of the European call option.
    """
    func = lambda sigma: black_scholes_call(S, K, T, r, sigma) - call_price
    implied_volatility = newton(func, x0=1)
    return implied_volatility

## Calculate implied volatilities for each call option
df_calls['calculated_implied_vol'] = df_calls.apply(
    lambda row: implied_volatility(
        S=AAPL_price,
        K=row['strike'],
        T=row['days_to_expire'] / 365.0,
        r=interest_rate,
        call_price=row['mid_price']
    ),
    axis=1
)

df_calls.head(2)
```

```
    current_date expiration_date type  strike_times_1000  highest_bid
305   2022-09-01      2022-09-30    C              70000         88.1  \
306   2022-09-01      2022-09-30    C              75000         83.1

     lowest_ask  implied_vol Ticker Symbol  days_to_expire  mid_price  strike
305        88.3     1.203935          AAPL              29       88.2    70.0  \
306        83.3     1.093580          AAPL              29       83.2    75.0

     calculated_implied_vol
305                1.211188
306                1.101416
```

(b) Plot the volatility smile for your call options.

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(8, 6))

## Sort the dataframe by strike price
df_calls_sorted = df_calls.sort_values(by='strike')

## Plot volatility smile
plt.plot(df_calls['strike'], df_calls['calculated_implied_vol'], color = 'green', marker='o', linestyle='-')
plt.xlabel('Strike Price')
plt.ylabel('Calculated Implied Volatility')
plt.title('Volatility Smile for Calls (22/09/01)')
plt.grid(True)
plt.show()
```

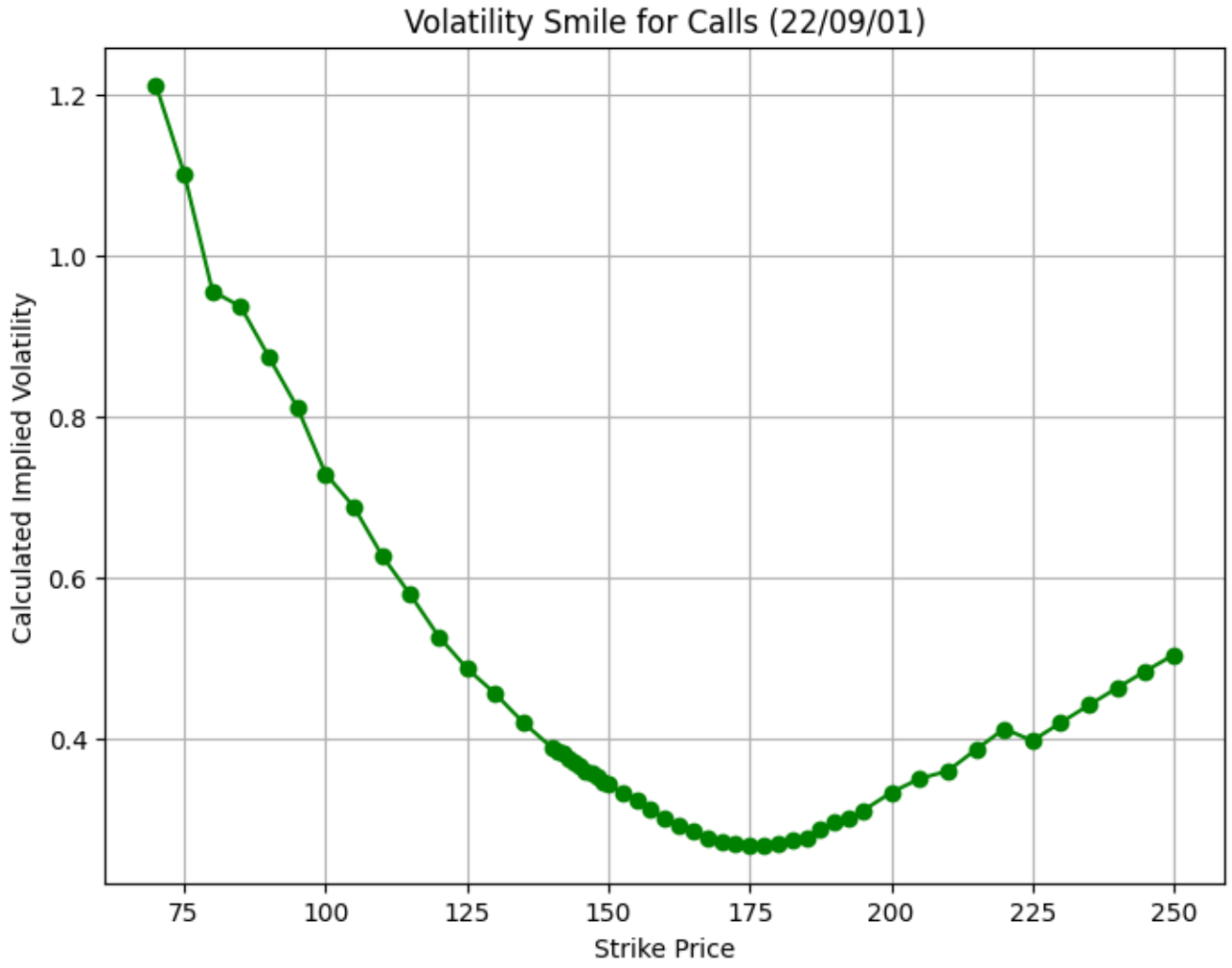
```
<Figure size 800x600 with 0 Axes>
```

```
[<matplotlib.lines.Line2D at 0x29aacd1d0>]
```

```
Text(0.5, 0, 'Strike Price')
```

```
Text(0, 0.5, 'Calculated Implied Volatility')
```

```
Text(0.5, 1.0, 'Volatility Smile for Calls (22/09/01)')
```



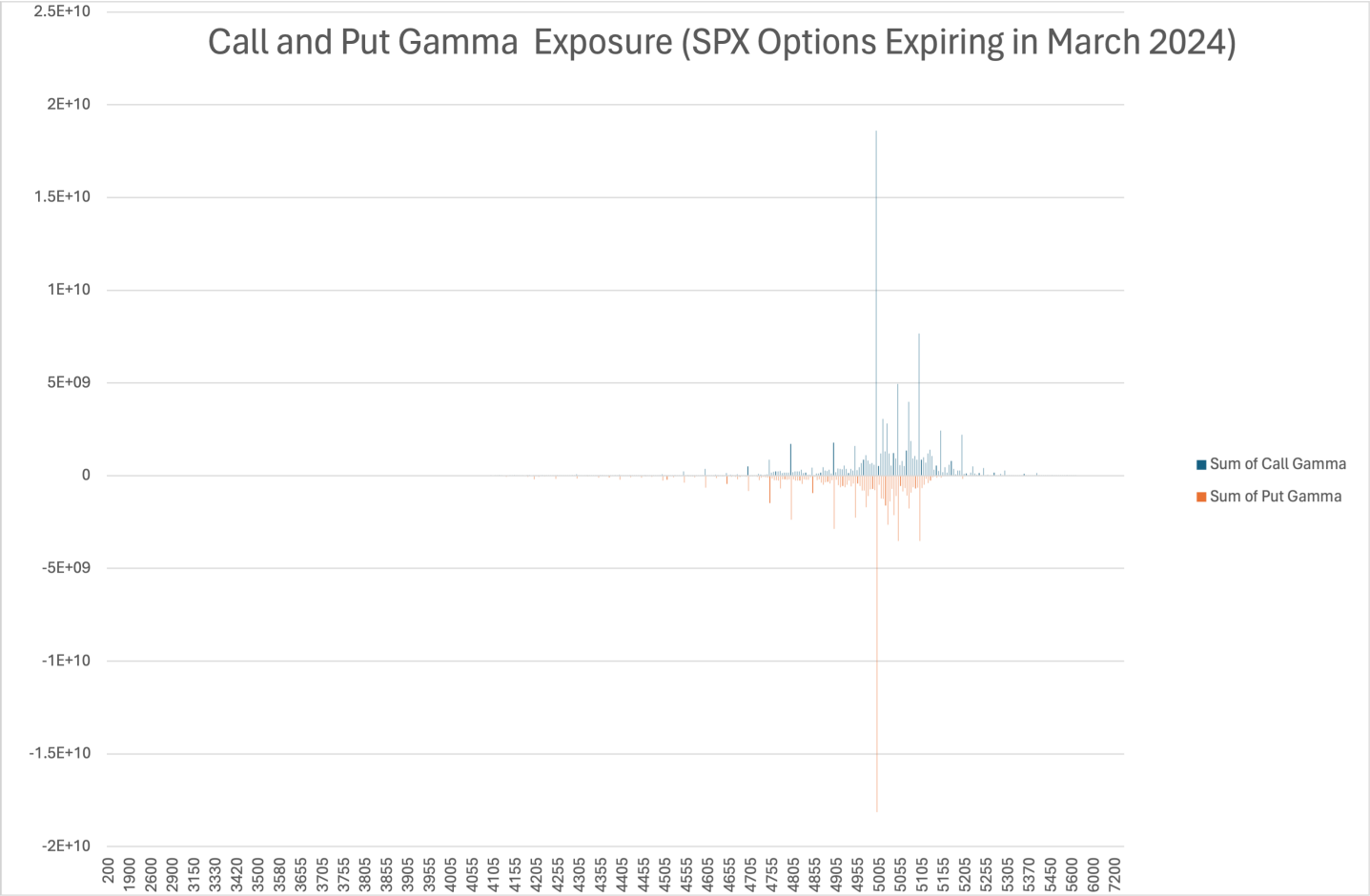
(c) Do you see a connection between features of the risk-neutral price distribution you found in the previous problem and the shape of the smile?

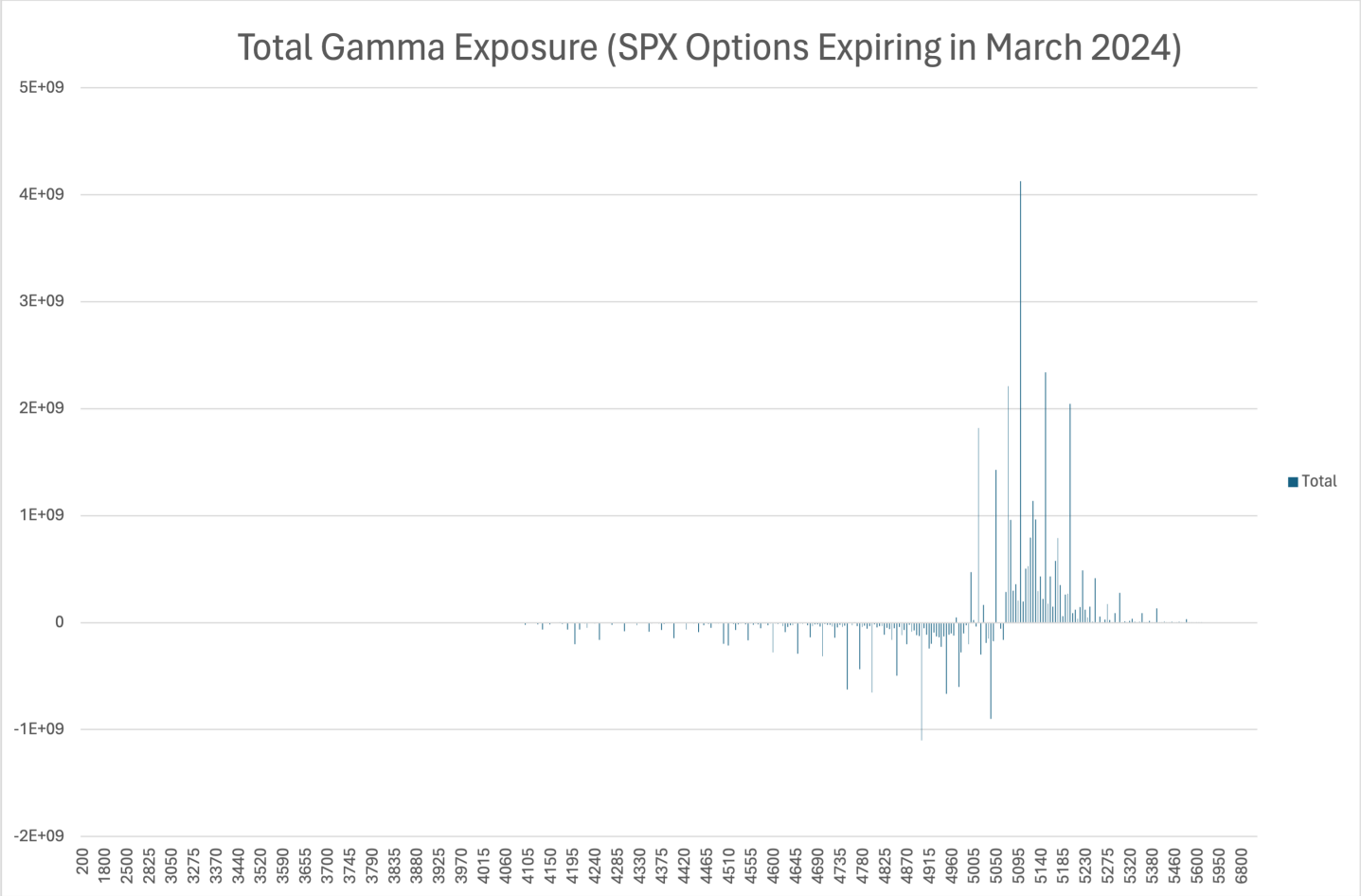
Answer Both the volatility smile and the risk-neutral price distribution share a left skew.

Exercise 4

Answer Because there were 10859 rows of Feb 28 2024 SPX options data across all expiration dates and strikes, we decided to limit our analysis to SPX options expiring in March 2024. The simple spot gamma charts, gamma profile charts and flip gamma are displayed below. (Related graphs and excel sheet are also submitted on Canvas.)

On Gamma exposure for options expiring in March, the total call exposure is 96898663586, and the total put exposure is -83726345904. This results in a net total exposure of 13172317681. We interpret this as option dealers need to BUY $13Bn worth of $SPX index for each 1% move DOWN, and SELL $19Bn for each 1% move UP.






```
# Read in quotedata and store it in a DataFrame called df.
index_name = "SPX"
filename = f"{index_name.lower()}_quotedata.csv"
df = pd.read_csv(filename, skiprows=3)

# Convert the column names to snake case and replace '.1' with '_puts'
df.columns = df.columns.str.lower().str.replace('.1', '_puts').str.replace(' ', '_')

# Print the column names of df
print(df.columns)
```

```
Index(['expiration_date', 'calls', 'last_sale', 'net', 'bid', 'ask', 'volume',
       'iv', 'delta', 'gamma', 'open_interest', 'strike', 'puts',
       'last_sale_puts', 'net_puts', 'bid_puts', 'ask_puts', 'volume_puts',
       'iv_puts', 'delta_puts', 'gamma_puts', 'open_interest_puts'],
      dtype='object')
```

```
# Read in the first row of the CSV to get the recent print
recent_price_df = pd.read_csv(filename, nrows=1, header=None)

# Get row 0 column 1
recent_price = recent_price_df.iloc[0, 1]
recent_price = float(recent_price.split(' ')[1])

print(recent_price)
```

```
5069.7598
```

```
import scipy.stats as stats

norm = stats.norm(loc=0, scale=1)
```

```
# Define a function to calculate the time to expiration in years
def calc_time_to_expiration(expiration_date_string):
    expiration_date = pd.to_datetime(expiration_date_string)
    current_date = pd.to_datetime("today")
    time_to_expiration = (expiration_date - current_date).days / 365
    return time_to_expiration


# Define a function to calculate gamma using black scholes
def calc_gamma(S, K, T, r, D, sigma, open_interest, option_type):
    if T == 0 or sigma == 0:
        return 0

    d1 = (np.log(S / K) + (r - D + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    gamma = np.exp(-D * T) * norm.pdf(d1) / (S * sigma * np.sqrt(T))

    if option_type == "call":
        return open_interest * 100 * S * S * 0.01 * gamma / (10**9)
    else:
        return open_interest * 100 * S * S * 0.01 * gamma * -1 / (10**9)


# Define a function that calculates the gammas exposure of a call or a put
def calc_option(strike_price, expiration_date, option_type, spots, df):
    # Get the time to expiration in years
    T = calc_time_to_expiration(expiration_date)

    # Get the row
    row = df[df["strike"] == strike_price]

    # Calc the gammas
    if option_type == "call":
        gamma_exposure = [
            calc_gamma(
                spot,
                strike_price,
                T,
                0.01,
                0.01,
                row["iv"].values[0],
                row["open_interest"].values[0],
                "call",
            )
            for spot in spots
        ]

        return gamma_exposure

    else:
        gamma_exposure = [
            calc_gamma(
                spot,
                strike_price,
                T,
                0.01,
                0.01,
                row["iv_puts"].values[0],
                row["open_interest_puts"].values[0],
```

```
                "put",
            )
            for spot in spots
        ]

        return gamma_exposure

# Set the desired strike
put_strike = 4900

min = 4200
max = 5800
step = put_strike / 1000

# Let's define an array of spot prices to look at
spots = np.arange(min, max, step)

# Set desired expiration date
expiration_date = "Thu Mar 28 2024"

# Calculate the put gammas
put_gamma_exposure = calc_option(put_strike, expiration_date, "put", spots, df)

# Plot the unit put gammas
plt.figure(figsize=(8, 6))
plt.plot(spots, put_gamma_exposure, color="blue")
plt.title(f"{index_name} Put Gamma Profile - {expiration_date}")
plt.xlabel("Spot Price")
plt.xlim(min, max)
plt.ylabel("Gamma Exposure ($ Billions per 1% Move in Spot Price)")
plt.grid(True)
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

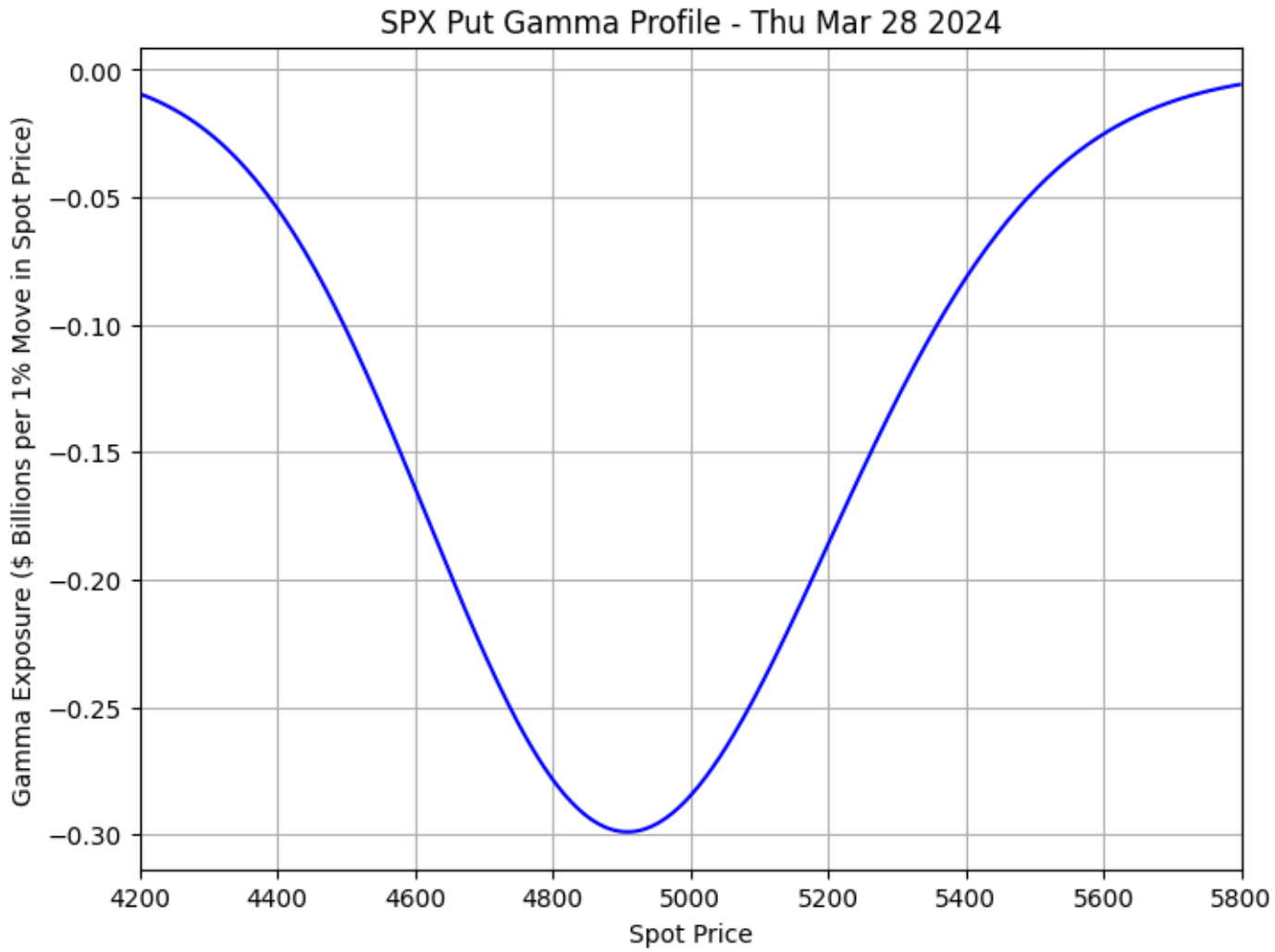
```
[<matplotlib.lines.Line2D at 0x29aa88250>]
```

```
Text(0.5, 1.0, 'SPX Put Gamma Profile - Thu Mar 28 2024')
```

```
Text(0.5, 0, 'Spot Price')
```

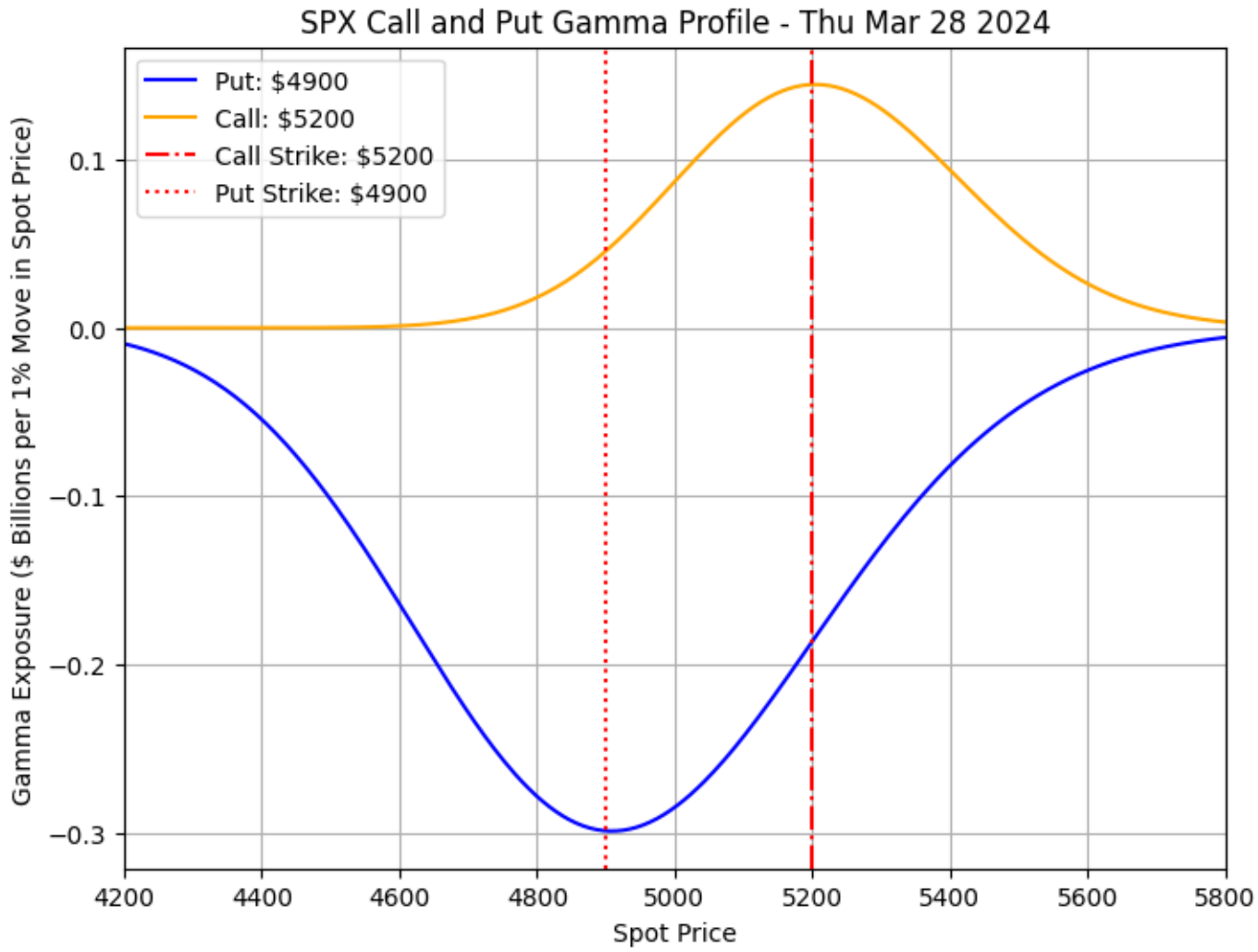
```
(4200.0, 5800.0)
```

```
Text(0, 0.5, 'Gamma Exposure ($ Billions per 1% Move in Spot Price)')
```




```
# Now let's add a call
call_strike = 5200

# Calculate the call gammas
call_gamma_exposure = calc_option(call_strike, expiration_date, "call", spots, df)

# Plot both the call and put gammas
plt.figure(figsize=(8, 6))

plt.plot(spots, put_gamma_exposure, label=f"Put: ${put_strike}", color="blue")
plt.plot(spots, call_gamma_exposure, label=f"Call: ${call_strike}", color="orange")

## Add vertical lines for call and put strikes
plt.axvline(x=call_strike, color='red', linestyle='-.', label=f'Call Strike: ${call_strike}')
plt.axvline(x=put_strike, color='red', linestyle=':', label=f'Put Strike: ${put_strike}')


plt.title(f"{index_name} Call and Put Gamma Profile - {expiration_date}")
plt.xlabel("Spot Price")
plt.xlim(min, max)


plt.ylabel("Gamma Exposure ($ Billions per 1% Move in Spot Price)")
plt.legend()
plt.grid(True)
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
[<matplotlib.lines.Line2D at 0x29ae8f910>]
[<matplotlib.lines.Line2D at 0x29ae578d0>]
<matplotlib.lines.Line2D at 0x29aac7290>
<matplotlib.lines.Line2D at 0x29b441390>
Text(0.5, 1.0, 'SPX Call and Put Gamma Profile - Thu Mar 28 2024')
Text(0.5, 0, 'Spot Price')
(4200.0, 5800.0)
Text(0, 0.5, 'Gamma Exposure ($ Billions per 1% Move in Spot Price)')
<matplotlib.legend.Legend at 0x29ae94310>
```




```python
# Add the gammas to get aggregate gamma exposure
aggregate_gamma_exposure = [
    put_gamma + call_gamma
    for put_gamma, call_gamma in zip(put_gamma_exposure, call_gamma_exposure)
]

# Plot all three
plt.figure(figsize=(8, 6))
plt.plot(spots, put_gamma_exposure, label=f"Put: ${put_strike}", alpha=0.5)
plt.plot(spots, call_gamma_exposure, label=f"Call: ${call_strike}", alpha=0.5)
plt.plot(spots, aggregate_gamma_exposure, label="Aggregate", color="green")
```

```
plt.title(f"{index_name} Aggregate Gamma Profile - {expiration_date}")
plt.xlabel("Spot Price")
plt.xlim(min, max)
plt.ylabel("Gamma Exposure ($ Billions per 1% Move in Spot Price)")
plt.legend()
plt.grid(True)
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

```
[<matplotlib.lines.Line2D at 0x29ae97a10>]
```

```
[<matplotlib.lines.Line2D at 0x29b4c6050>]
```

```
[<matplotlib.lines.Line2D at 0x29b4c6010>]
```

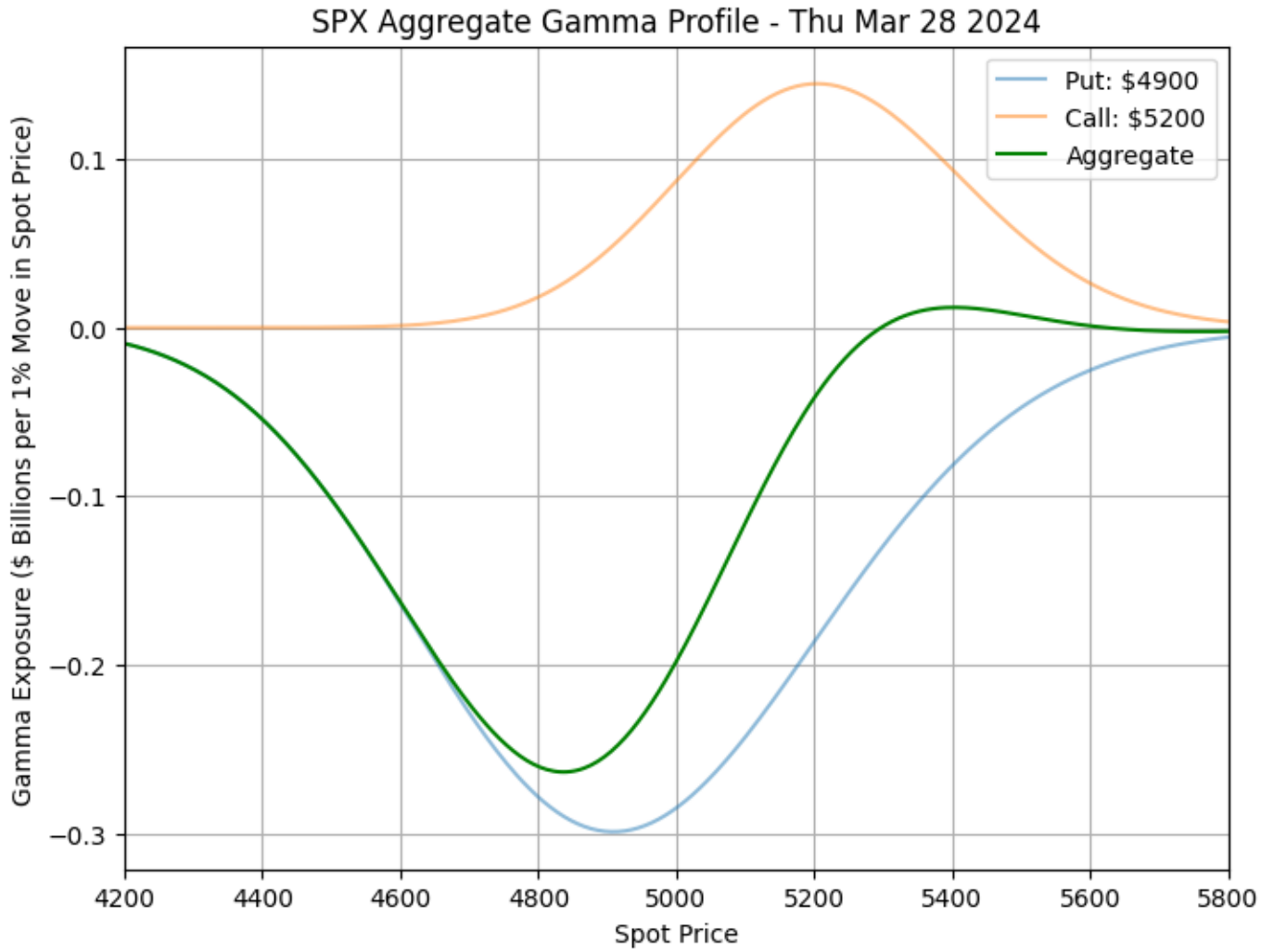
```
Text(0.5, 1.0, 'SPX Aggregate Gamma Profile - Thu Mar 28 2024')
```

```
Text(0.5, 0, 'Spot Price')
```

```
(4200.0, 5800.0)
```

```
Text(0, 0.5, 'Gamma Exposure ($ Billions per 1% Move in Spot Price)')
```

```
<matplotlib.legend.Legend at 0x29b44fbd0>
```




```
# Let's aggregate more strikes

put_strikes = [4900, 5000]
call_strikes = [5100, 5200]
```

```python
# Calculate the put gammas
put_gamma_exposures = [
    calc_option(strike, expiration_date, "put", spots, df) for strike in put_strikes
]

# Calculate the call gammas
call_gamma_exposures = [
    calc_option(strike, expiration_date, "call", spots, df) for strike in call_strikes
]

# Add the gammas to get aggregate gamma exposure
aggregate_gamma_exposure = [
    sum(gammas) for gammas in zip(*put_gamma_exposures, *call_gamma_exposures)
]

# Plotting
plt.figure(figsize=(8, 6))

for put_gamma_exposure, put_strike in zip(put_gamma_exposures, put_strikes):
    plt.plot(spots, put_gamma_exposure, label=f"Put: ${put_strike}", alpha=0.5)

for call_gamma_exposure, call_strike in zip(call_gamma_exposures, call_strikes):
    plt.plot(spots, call_gamma_exposure, label=f"Call: ${call_strike}", alpha=0.5)

plt.plot(spots, aggregate_gamma_exposure, label="Aggregate", color="green")

plt.title(f"{index_name} Aggregate Gamma Profile - {expiration_date}")
plt.xlabel("Spot Price")
plt.xlim(min, max)
plt.ylabel("Gamma Exposure ($ Billions per 1% Move in Spot Price)")
plt.legend()
plt.grid(True)
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

```
[<matplotlib.lines.Line2D at 0x2a38a5990>]
```

```
[<matplotlib.lines.Line2D at 0x2a38a61d0>]
```

```
[<matplotlib.lines.Line2D at 0x2a38a7610>]
```

```
[<matplotlib.lines.Line2D at 0x2a3866210>]
```

```
[<matplotlib.lines.Line2D at 0x2a466a910>]
```

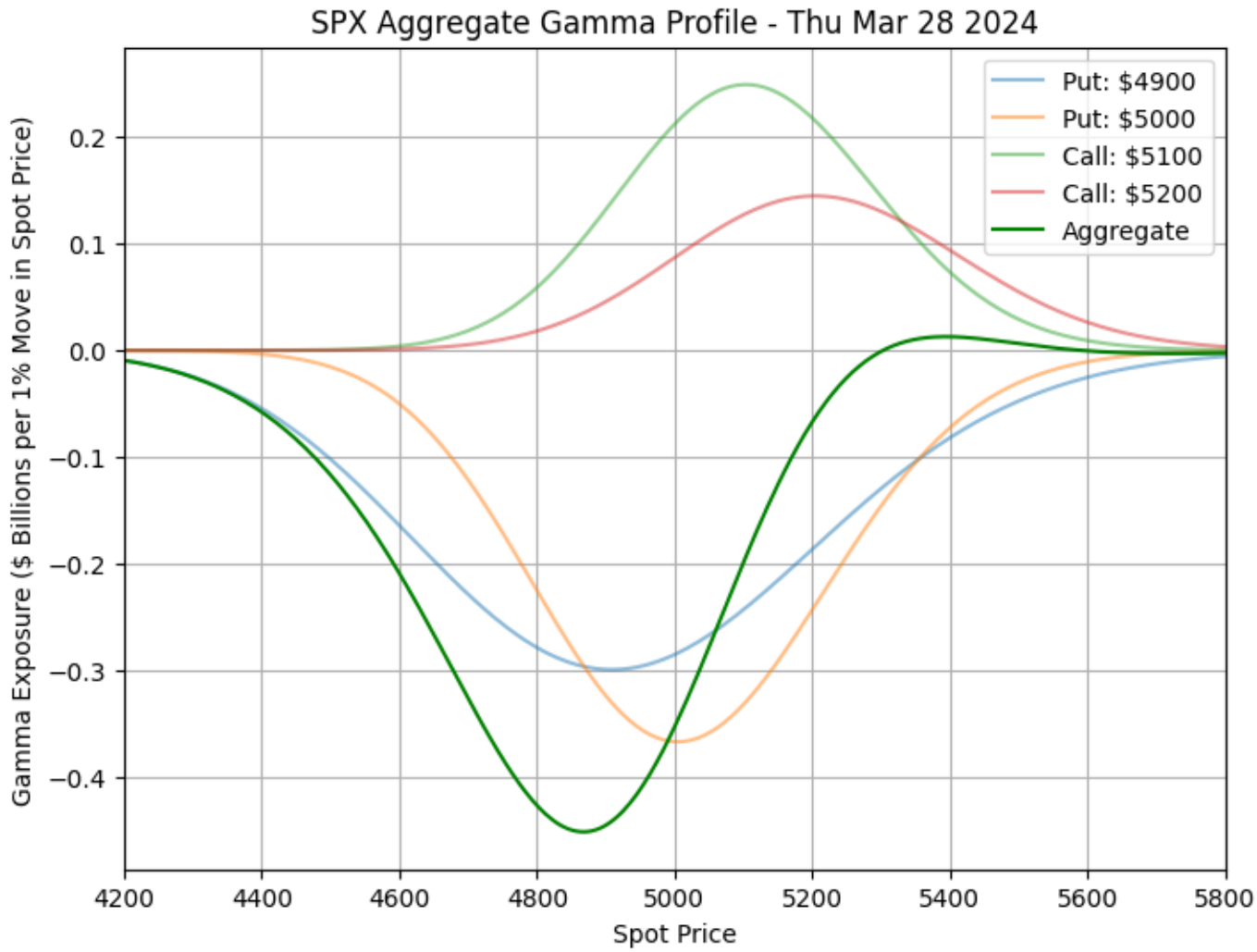
```
Text(0.5, 1.0, 'SPX Aggregate Gamma Profile - Thu Mar 28 2024')
```

```
Text(0.5, 0, 'Spot Price')
```

```
(4200.0, 5800.0)
```

```
Text(0, 0.5, 'Gamma Exposure ($ Billions per 1% Move in Spot Price)')
```

```
<matplotlib.legend.Legend at 0x2a3881590>
```




```
%%time

# Now let's plot all gamma exposures for all march expiration dates
put_gamma_exposures = []
call_gamma_exposures = []

spots = np.linspace(4000, 6000, 1000)
min = 4000
max = 6000

for index, row in df.iterrows():
    expiration_date = row["expiration_date"]

    put_gamma = calc_option(row["strike"], expiration_date, "put", spots, df)
    put_gamma_exposures.append(put_gamma)

    call_gamma = calc_option(row["strike"], expiration_date, "call", spots, df)
    call_gamma_exposures.append(call_gamma)

aggregate_gamma_exposure = [
    sum(gammas) for gammas in zip(*put_gamma_exposures, *call_gamma_exposures)
]

# Where aggregate_gamma_exposure changes sign
gamma_flip = [
    i
```

```
    for i in range(1, len(aggregate_gamma_exposure))
    if np.sign(aggregate_gamma_exposure[i]) != np.sign(aggregate_gamma_exposure[i - 1])
]
```

```
CPU times: user 4min 34s, sys: 1.43 s, total: 4min 36s
Wall time: 4min 37s
```

```
# Plotting
plt.figure(figsize=(8, 6))

plt.plot(spots, aggregate_gamma_exposure, label="Aggregate", color="green")

# Vertical line at recent price
plt.axvline(
    x=recent_price, color="red", linestyle="--", label=f"Recent Price: {recent_price}"
)

# Plot the gamma flip
for i in gamma_flip:
    plt.axvline(
        x=spots[i], color="black", linestyle=":", label=f"Gamma Flip: {spots[i]}"
    )

plt.title(f"{index_name} Aggregate Gamma Profile - All March Expiration Dates")
plt.xlabel("Spot Price")
plt.xlim(min, max)
plt.ylabel("Gamma Exposure ($ Billions per 1% Move in Spot Price)")
plt.grid(True)
plt.legend()
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

```
[<matplotlib.lines.Line2D at 0x2aac47b90>]
```

```
<matplotlib.lines.Line2D at 0x29b487e10>
```

```
<matplotlib.lines.Line2D at 0x2aac458d0>
```

```
<matplotlib.lines.Line2D at 0x2996fc290>
```

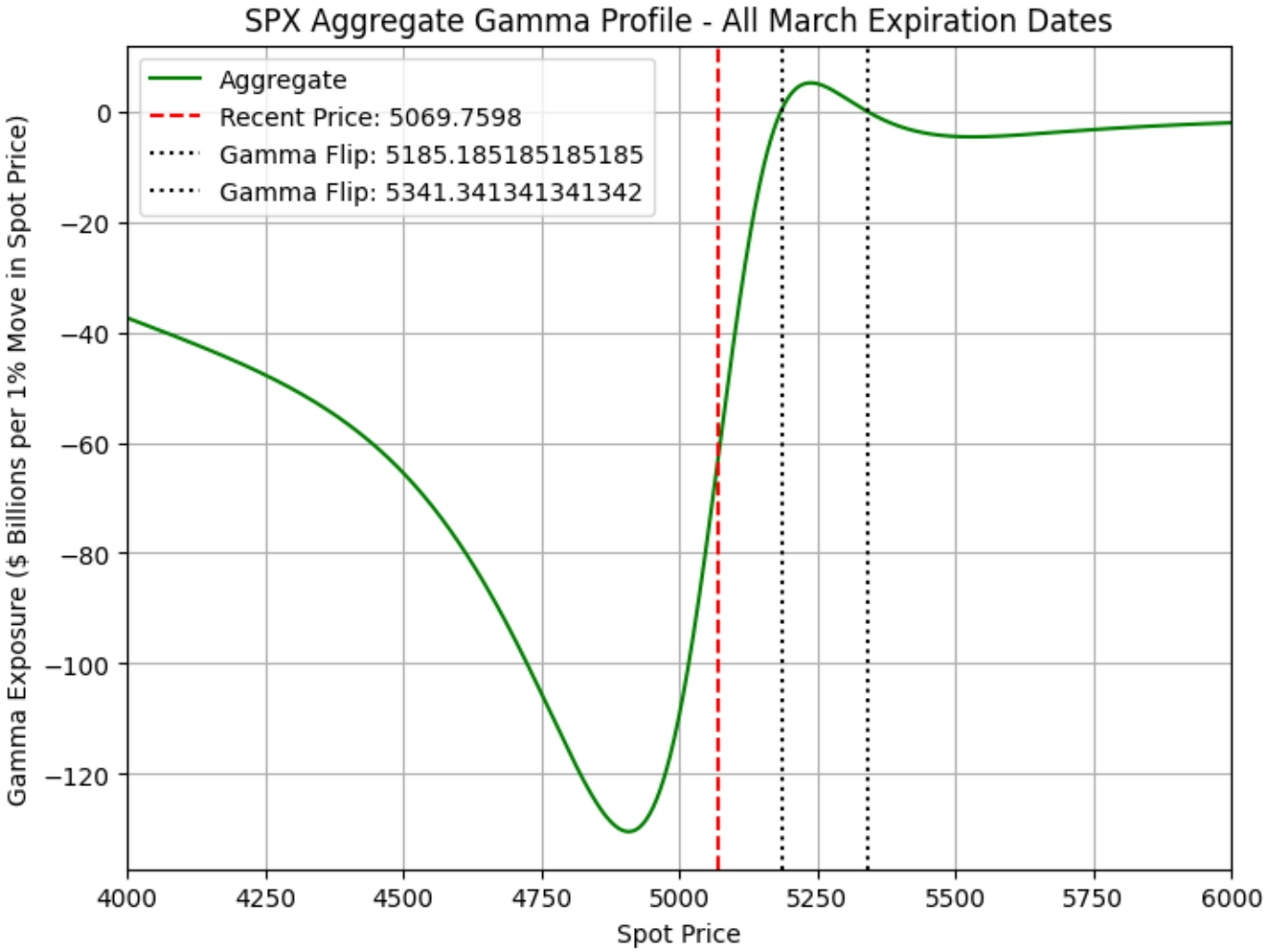
```
Text(0.5, 1.0, 'SPX Aggregate Gamma Profile - All March Expiration Dates')
```

```
Text(0.5, 0, 'Spot Price')
```

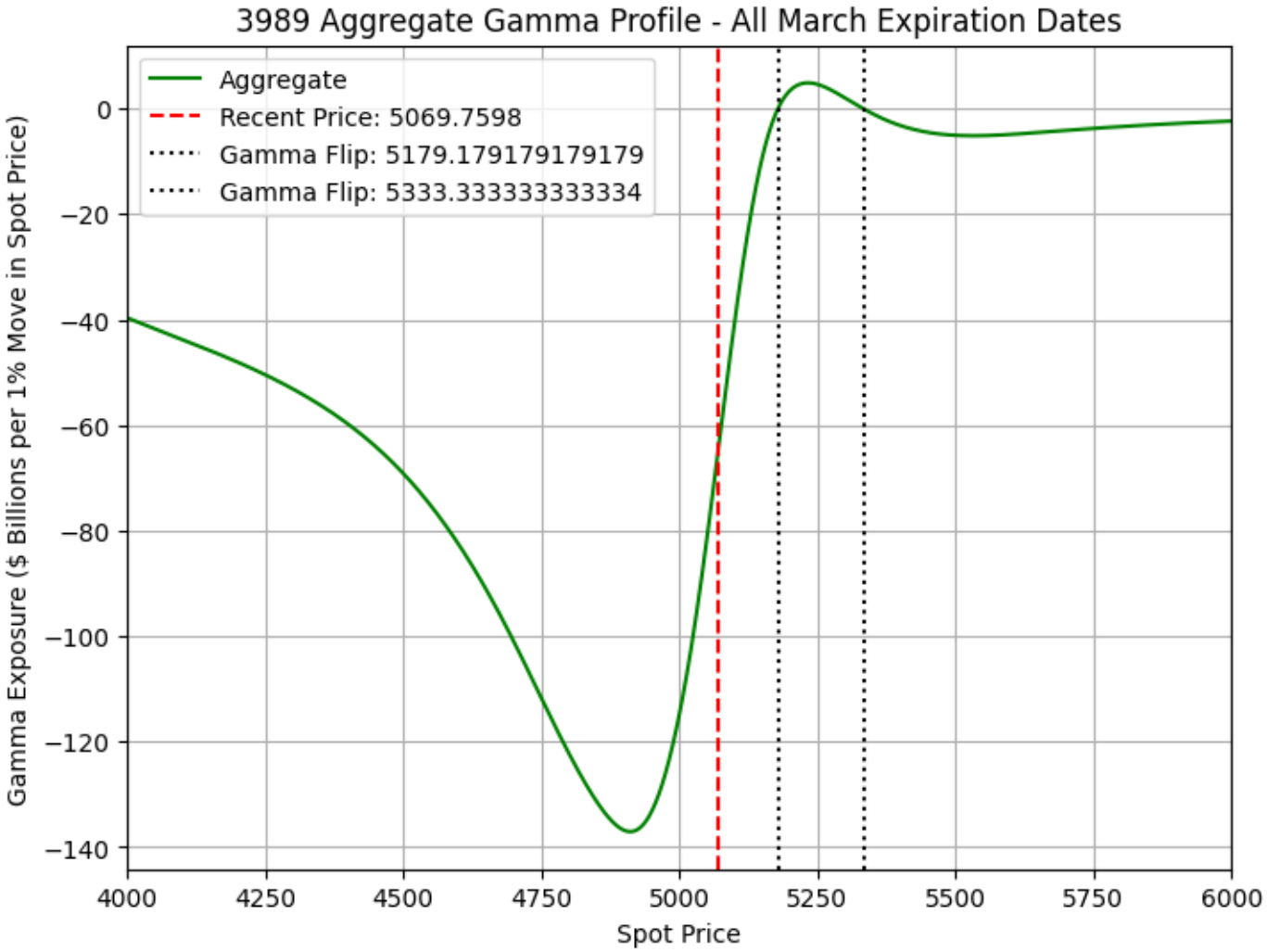
```
(4000.0, 6000.0)
```

```
Text(0, 0.5, 'Gamma Exposure ($ Billions per 1% Move in Spot Price)')
```

```
<matplotlib.legend.Legend at 0x299320910>
```



In case the code above does not run:


3989 Aggregate Gamma Profile - All March Expiration Dates
Aggregate
Recent Price: 5069.7598
Gamma Flip: 5179.179179179179
Gamma Flip: 5333.333333333334
Gamma Exposure ($ Billions per 1% Move in Spot Price)
Spot Price
0
−20
−40
−60
−80
−100
−120
−140
4000
4250
4500
4750
5000
5250
5500
5750
6000