In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Data Section

Load First Trading Day Data

Load the dataset corresponding to the first trading day (DATA1).
Ensure the data is properly formatted and cleaned for subsequent analysis and processing.

In [4]:
```
import os,math
import pandas as pd
from datetime import datetime
from scipy.stats import norm

BASE = "/Users/simratkaurrandhawa/Desktop/Sem 4/fe621/Assignments/DATA1"

meta = pd.read_csv(os.path.join(BASE, "META.csv"))
snap = pd.read_csv(os.path.join(BASE, "SNAP_TSLA_SPY.csv"))

tsla_opt = pd.read_csv(os.path.join(BASE, "TSLA_US_Equity_OPTIONS_NEXT3MONTHS_
spy_opt  = pd.read_csv(os.path.join(BASE, "SPY_US_Equity_OPTIONS_NEXT3MONTHS_3

asof_date = datetime.strptime(meta.loc[0, "asof_date"], "%Y-%m-%d").date()

S0_tsla = float(snap.loc[0, "PX_LAST"])
S0_spy  = float(snap.loc[1, "PX_LAST"])
```

In [5]:
```
r = 0.0364   # effective Fed funds rate around Feb 6
```

In [6]:
```
tsla_opt.head()
```

Out[6]:

	PX_LAST	BID	ASK	VOLUME	OPEN_INT	IVOL_MID	DELTA	GAMMA	THETA	
0	313.33	312.00	313.80	300	4133	249.8019	1.000000	0.000000	NaN	0.00
1	NaN	300.70	304.55	0	22	233.8409	0.990314	0.000393	NaN	0.02
2	319.63	290.75	294.60	90	141	216.6500	0.989626	0.000446	NaN	0.0
3	309.87	280.75	284.60	5	43	207.9268	0.988637	0.000512	NaN	0.02
4	299.78	270.75	274.60	6	29	188.6663	0.987963	0.000574	NaN	0.02

In [7]:
```
spy_opt.head()
```

Out[7]:

	PX_LAST	BID	ASK	VOLUME	OPEN_INT	IVOL_MID	DELTA	GAMMA	THETA	
0	362.17	353.42	356.16	1	3	109.06250	0.985944	0.000999	NaN	0.0
1	NaN	348.39	351.16	0	0	109.84220	0.999765	0.000042	NaN	0.0
2	348.01	343.42	346.17	2	2	109.83110	0.985287	0.001077	NaN	0.0
3	337.41	338.45	341.19	7	12	117.17140	1.000000	0.000000	NaN	0.0
4	NaN	333.40	336.20	0	0	95.07065	0.999919	0.000018	NaN	0.0

Helper Functions

In this section, we implement the main functions needed for pricing options and computing implied volatility.

- `black_scholes`
  Computes the theoretical price of a European call or put option using the Black–Scholes formula.

- `bisection`
  Implements the Bisection method to numerically find the root of a function.
  We use this to solve for implied volatility.

- `implied_vol_row`
  Calculates implied volatility for a single option using the Bisection method by matching the Black–Scholes price to the observed market price.

- `implied_vol_newton`
  Computes implied volatility using the Newton–Raphson method.
  This approach is typically faster but depends on a good initial guess and stable Vega values.

In [8]:

```
def black_scholes(S0, K, T ,r, sigma, option_type):
    d1 = (math.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * math.sqrt(T)
    d2 = d1 - sigma * math.sqrt(T)

    if option_type.lower() == "call":
        price =  S0 * norm.cdf(d1) - K * math.exp(-r * T) * norm.cdf(d2)
    else:  # put
        price = K * math.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)

    return price, d1
```

In [9]:

```
def bisection(f, a, b, tol=1e-6, max_iter=200):
    fa, fb = f(a), f(b)
    if np.isnan(fa) or np.isnan(fb) or fa * fb > 0:
```

```python
        return np.nan

    for _ in range(max_iter):
        m  = 0.5 * (a + b)
        fm = f(m)

        if np.isnan(fm):
            return np.nan
        if abs(fm) < tol or (b - a) / 2 < tol:
            return m

        if fa * fm < 0:
            b, fb = m, fm
        else:
            a, fa = m, fm

    return m
```

In [10]:
```python
def implied_vol_row(S, K, T, r, mid, cp):
    def f(sig):
        price, d1 = black_sholes(S, K, T, r, sig, cp)
        return price - mid

    low, high = 1e-6, 5.0
    fl, fh = f(low), f(high)

    tries = 0
    while (not np.isnan(fl) and not np.isnan(fh)) and fl * fh > 0 and high < 1(
        high *= 2
        fh = f(high)
        tries += 1

    return bisection(f, low, high, tol=1e-6)
```

In [11]:
```python
def implied_vol_newton(S, K, T, r, mid, cp, x0=0.2, tol=1e-6, max_iter=100):
    sigma = max(x0, 1e-6)

    for _ in range(max_iter):
        price, d1 = black_sholes(S, K, T, r, sigma, cp)
        f = price - mid
        if abs(f) < tol:
            return sigma

        vega = S * norm.pdf(d1) * np.sqrt(T)

        if(not np.isfinite(vega)) or vega < 1e-10:
            return np.nan

        sigma_new = sigma - f /vega
        if(not np.isfinite(sigma_new)) or sigma_new <= 0:
            return np.nan

        sigma = sigma_new

    return np.nan
```

Applying the Functions to TSLA and SPY Data

In this section, we apply the helper functions to the option data for TSLA and SPY (DATA1).

For each option:

- Use the mid-price (average of bid and ask) as the market price.
- Compute time to maturity in years.
- Use the chosen short-term interest rate.
- Apply `implied_vol_row` (Bisection) to compute implied volatility.
- Optionally compare results using `implied_vol_newton`.

The resulting implied volatilities are then added to the dataset for further analysis and comparison bet

In [12]:
```
df_tsla = tsla_opt.copy()

df_tsla = df_tsla[df_tsla['VOLUME'].fillna(0) > 0].copy()
df_tsla = df_tsla.dropna(subset=['BID', 'ASK', 'OPT_STRIKE_PX', 'MATURITY', 'O

df_tsla['MID'] = 0.5 * (df_tsla['BID'] + df_tsla['ASK'])
df_tsla['K'] = df_tsla['OPT_STRIKE_PX'].astype(float)
df_tsla['MATURITY'] = pd.to_datetime(df_tsla['MATURITY']).dt.date

df_tsla['T'] = df_tsla['MATURITY'].apply(lambda d: (d - asof_date).days / 365.
```

In [13]:
```
df_tsla.head()
```

Out[13]:

	PX_LAST	BID	ASK	VOLUME	OPEN_INT	IVOL_MID	DELTA	GAMMA	THETA	V
0	313.33	312.00	313.8	300	4133	249.8019	1.000000	0.000000	NaN	0.00(
2	319.63	290.75	294.6	90	141	216.6500	0.989626	0.000446	NaN	0.02
3	309.87	280.75	284.6	5	43	207.9268	0.988637	0.000512	NaN	0.02
4	299.78	270.75	274.6	6	29	188.6663	0.987963	0.000574	NaN	0.02
5	245.82	260.85	264.5	4	115	181.7922	1.000000	0.000000	NaN	0.00(

In [14]:
```
df_tsla = df_tsla[df_tsla['T'] > 0].copy()
```

In [15]:
```
df_tsla['IV'] = df_tsla.apply(
    lambda row: implied_vol_row(S0_tsla, row['K'], row['T'], r, row['MID'], ro
    axis=1
)
```

In [16]:
```
df_tsla['IV'].describe()
```

Out[16]:
```
count    746.000000
mean       0.679609
std        0.355561
min        0.401609
25%        0.461229
50%        0.568171
75%        0.783671
max        5.079289
Name: IV, dtype: float64
```

In [17]:
```
df_tsla['IV_newton'] = df_tsla.apply(
    lambda row: implied_vol_newton(S0_tsla, row['K'], row['T'], r, row['MID'],
    axis=1
)
```

In [18]:
```
df_tsla['IV_newton'].describe()
```

Out[18]:
```
count    746.000000
mean       0.679609
std        0.355561
min        0.401609
25%        0.461229
50%        0.568171
75%        0.783671
max        5.079289
Name: IV_newton, dtype: float64
```

In [19]:
```
df_spy = spy_opt.copy()

df_spy = df_spy[df_spy['VOLUME'].fillna(0) > 0].copy()
df_spy = df_spy.dropna(subset=['BID', 'ASK', 'OPT_STRIKE_PX', 'MATURITY', 'OPT_

df_spy['MID'] = 0.5 * (df_spy['BID'] + df_spy['ASK'])
df_spy['K'] = df_spy['OPT_STRIKE_PX'].astype(float)
df_spy['MATURITY'] = pd.to_datetime(df_spy['MATURITY']).dt.date

df_spy['T'] = df_spy['MATURITY'].apply(lambda d: (d - asof_date).days / 365.0)
```

In [20]:
```
df_spy.head()
```

Out[20]:

	PX_LAST	BID	ASK	VOLUME	OPEN_INT	IVOL_MID	DELTA	GAMMA	THETA	
0	362.17	353.42	356.16	1	3	109.0625	0.985944	0.000999	NaN	0.04
2	348.01	343.42	346.17	2	2	109.8311	0.985287	0.001077	NaN	0.04
3	337.41	338.45	341.19	7	12	117.1714	1.000000	0.000000	NaN	0.00
8	314.36	313.48	316.24	1	22	102.7098	0.998946	0.000172	NaN	0.00
9	308.53	308.45	311.23	4	4	102.9566	0.996619	0.000444	NaN	0.0

In [21]:
```
df_spy = df_spy[df_spy['T'] > 0].copy()

df_spy['IV'] = df_spy.apply(
```

```
    lambda row: implied_vol_row(S0_spy, row['K'], row['T'], r, row['MID'], row
    axis=1
)

df_spy['IV_newton'] = df_spy.apply(
    lambda row: implied_vol_newton(S0_spy, row['K'], row['T'], r, row['MID'],
    axis=1
)
```

Put–Call Parity

Here we check whether the option prices satisfy the Put–Call Parity relationship:

C – P = S – K e^(–rT)

For TSLA and SPY, we match calls and puts with the same strike and maturity, use mid-prices, and compare both sides of the equation.

If there are large differences, it may suggest pricing inconsistencies or possible arbitrage opportunities.

In [22]:
```
calls = df_tsla[df_tsla['OPT_PUT_CALL'] == 'Call'].copy()
puts = df_tsla[df_tsla['OPT_PUT_CALL'] == 'Put'].copy()

df_parity = calls.merge(puts, on=['K', 'MATURITY'], suffixes=('_call','_put'))
```

In [23]:
```
df_parity['discount'] = np.exp(-r * df_parity['T_call'])

df_parity['C_from_P'] = (df_parity['MID_put'] + S0_tsla - df_parity['K'] * df_
df_parity['P_from_C'] = (df_parity['MID_call']- S0_tsla + df_parity['K'] * df_
```

In [24]:
```
df_parity['call_diff'] = df_parity['C_from_P'] - df_parity['MID_call']
df_parity['put_diff']  = df_parity['P_from_C'] - df_parity['MID_put']
```

In [25]:
```
df_parity[['K','MATURITY','call_diff','put_diff']].describe()
```

Out[25]:

	K	call_diff	put_diff
count	335.000000	335.000000	335.000000
mean	414.604478	0.718804	-0.718804
std	165.226358	1.261175	1.261175
min	100.000000	-0.398866	-6.644122
25%	282.500000	-0.078214	-0.971066
50%	420.000000	0.103918	-0.103918
75%	542.500000	0.971066	0.078214
max	940.000000	6.644122	0.398866

In [26]:
```
calls_spy = df_spy[df_spy['OPT_PUT_CALL'] == 'Call'].copy()
puts_spy = df_spy[df_spy['OPT_PUT_CALL'] == 'Put'].copy()
```

```
df_parity_spy = calls_spy.merge(puts_spy, on=['K', 'MATURITY'], suffixes=('_ca

df_parity_spy['discount'] = np.exp(-r * df_parity_spy['T_call'])

df_parity_spy['C_from_P'] = (df_parity_spy['MID_put'] + S0_spy - df_parity_spy
df_parity_spy['P_from_C'] = (df_parity_spy['MID_call']- S0_spy + df_parity_spy

df_parity_spy['call_diff'] = df_parity_spy['C_from_P'] - df_parity_spy['MID_ca
df_parity_spy['put_diff']  = df_parity_spy['P_from_C'] - df_parity_spy['MID_pu
```

In [27]:
```
df_parity_spy[['K','MATURITY','call_diff','put_diff']].describe()
```

Out[27]:

	K	call_diff	put_diff
count	443.000000	443.000000	443.000000
mean	619.498871	0.837560	-0.837560
std	118.528256	1.113645	1.113645
min	245.000000	-0.095917	-5.714123
25%	555.000000	-0.019185	-1.375103
50%	662.000000	0.436399	-0.436399
75%	698.500000	1.375103	0.019185
max	895.000000	5.714123	0.095917

Combining TSLA and SPY Data

In this section, we combine the TSLA and SPY option datasets into a single DataFrame.

This allows us to:

- Compare implied volatilities side by side
- Analyze differences in pricing behavior
- Summarize key statistics (mean IV, distribution, etc.)

By working with one combined dataset, we can more easily compare how volatility and option pricing differ between TSLA and SPY.

In [28]:
```
df_tsla = df_tsla.rename(columns={'IV': 'IV_bisect'})
df_spy  = df_spy.rename(columns={'IV': 'IV_bisect'})
```

In [29]:
```
df_tsla['STOCK'] = 'TSLA'
df_spy['STOCK']  = 'SPY'

df_tsla['S0'] = float(S0_tsla)
df_spy['S0']  = float(S0_spy)

df_all = pd.concat([df_tsla, df_spy], ignore_index=True)
```

In [30]:
```
df_all = df_all[['STOCK','MATURITY','OPT_PUT_CALL','K','T','IV_bisect','IV_new
```

In [31]:
```
df_all.head()
```

Out[31]:

	STOCK	MATURITY	OPT_PUT_CALL	K	T	IV_bisect	IV_newton	S0
0	TSLA	2026-02-20	Call	100.0	0.038356	2.659078	2.659078	412.66
1	TSLA	2026-02-20	Call	120.0	0.038356	NaN	NaN	412.66
2	TSLA	2026-02-20	Call	130.0	0.038356	NaN	NaN	412.66
3	TSLA	2026-02-20	Call	140.0	0.038356	NaN	NaN	412.66
4	TSLA	2026-02-20	Call	150.0	0.038356	NaN	NaN	412.66

In [32]:
```
df_all['abs_diff'] = (df_all['IV_newton'] - df_all['IV_bisect']).abs()

df_all[['IV_bisect','IV_newton','abs_diff']].describe()
```

Out[32]:

	IV_bisect	IV_newton	abs_diff
count	1770.000000	1770.000000	1.770000e+03
mean	0.451903	0.451903	1.787404e-07
std	0.338097	0.338097	1.857322e-07
min	0.097872	0.097873	0.000000e+00
25%	0.186650	0.186650	0.000000e+00
50%	0.429858	0.429858	1.249284e-07
75%	0.598653	0.598653	3.271623e-07
max	5.079289	5.079289	5.851142e-07

In [33]:
```
summary = (
    df_all
    .groupby(['STOCK','MATURITY','OPT_PUT_CALL'], as_index=False)
    .agg(
        avg_iv_bisect=('IV_bisect','mean'),
        avg_iv_newton=('IV_newton','mean'),
        n=('IV_bisect','count')
    )
    .sort_values(['STOCK','MATURITY','OPT_PUT_CALL'])
)

summary
```

Out[33]:

	STOCK	MATURITY	OPT_PUT_CALL	avg_iv_bisect	avg_iv_newton	n
0	SPY	2026-02-20	Call	0.322806	0.322806	162
1	SPY	2026-02-20	Put	0.294629	0.294629	172
2	SPY	2026-03-20	Call	0.296330	0.296330	186
3	SPY	2026-03-20	Put	0.331205	0.331205	203
4	SPY	2026-04-17	Call	0.152728	0.152728	145
5	SPY	2026-04-17	Put	0.291106	0.291106	156
6	TSLA	2026-02-20	Call	0.651255	0.651255	99
7	TSLA	2026-02-20	Put	0.777306	0.777306	115
8	TSLA	2026-03-20	Call	0.707327	0.707327	142
9	TSLA	2026-03-20	Put	0.690194	0.690194	119
10	TSLA	2026-04-17	Call	0.607804	0.607804	145
11	TSLA	2026-04-17	Put	0.654114	0.654114	126

In [34]:

```
df_all['IV_final'] = df_all['IV_newton'].fillna(df_all['IV_bisect'])

stock_compare = (
    df_all.dropna(subset=['IV_final'])
    .groupby('STOCK', as_index=False)
    .agg(avg_iv=('IV_final','mean'), med_iv=('IV_final','median'), n=('IV_fina
)
stock_compare
```

Out[34]:

	STOCK	avg_iv	med_iv	n
0	SPY	0.286017	0.202131	1024
1	TSLA	0.679609	0.568171	746

In [35]:

```
df_all['is_call'] = df_all['OPT_PUT_CALL'].str.contains('C')
df_all['ITM'] = np.where(df_all['is_call'], df_all['S0'] > df_all['K'], df_all
df_all['moneyness'] = df_all['K'] / df_all['S0']

# Simple 3-bucket (ATM within 2%)
df_all['bucket'] = np.where(
    np.abs(df_all['moneyness'] - 1.0) <= 0.02, 'ATM',
    np.where(df_all['ITM'], 'ITM', 'OTM')
)

bucket_table = (
    df_all.dropna(subset=['IV_final'])
    .groupby(['STOCK','MATURITY','OPT_PUT_CALL','bucket'], as_index=False)
    .agg(avg_iv=('IV_final','mean'), n=('IV_final','count'))
    .sort_values(['STOCK','MATURITY','OPT_PUT_CALL','bucket'])
)
bucket_table
```

Out[35]:

	STOCK	MATURITY	OPT_PUT_CALL	bucket	avg_iv	n
0	SPY	2026-02-20	Call	ATM	0.141979	28
1	SPY	2026-02-20	Call	ITM	0.438504	101
2	SPY	2026-02-20	Call	OTM	0.122130	33
3	SPY	2026-02-20	Put	ATM	0.143064	28
4	SPY	2026-02-20	Put	ITM	0.210820	32
5	SPY	2026-02-20	Put	OTM	0.356465	112
6	SPY	2026-03-20	Call	ATM	0.150823	28
7	SPY	2026-03-20	Call	ITM	0.418961	101
8	SPY	2026-03-20	Call	OTM	0.150515	57
9	SPY	2026-03-20	Put	ATM	0.158465	28
10	SPY	2026-03-20	Put	ITM	0.192929	36
11	SPY	2026-03-20	Put	OTM	0.401815	139
12	SPY	2026-04-17	Call	ATM	0.148157	28
13	SPY	2026-04-17	Call	ITM	0.197364	46
14	SPY	2026-04-17	Call	OTM	0.125612	71
15	SPY	2026-04-17	Put	ATM	0.162517	28
16	SPY	2026-04-17	Put	ITM	0.184791	32
17	SPY	2026-04-17	Put	OTM	0.364050	96
18	TSLA	2026-02-20	Call	ATM	0.411030	6
19	TSLA	2026-02-20	Call	ITM	0.884945	41
20	TSLA	2026-02-20	Call	OTM	0.494717	52
21	TSLA	2026-02-20	Put	ATM	0.414273	6
22	TSLA	2026-02-20	Put	ITM	0.817316	68
23	TSLA	2026-02-20	Put	OTM	0.764075	41
24	TSLA	2026-03-20	Call	ATM	0.435559	4
25	TSLA	2026-03-20	Call	ITM	0.940458	62
26	TSLA	2026-03-20	Call	OTM	0.531445	76
27	TSLA	2026-03-20	Put	ATM	0.435836	4
28	TSLA	2026-03-20	Put	ITM	0.622967	55
29	TSLA	2026-03-20	Put	OTM	0.768776	60
30	TSLA	2026-04-17	Call	ATM	0.448479	4
31	TSLA	2026-04-17	Call	ITM	0.755369	58
32	TSLA	2026-04-17	Call	OTM	0.512365	83
33	TSLA	2026-04-17	Put	ATM	0.448363	4
34	TSLA	2026-04-17	Put	ITM	0.584267	58

	STOCK	MATURITY	OPT_PUT_CALL	bucket	avg_iv	n
35	TSLA	2026-04-17	Put	OTM	0.730273	64

In [36]:
```
vix_level = 17.76   # VIX index on Feb 6, 2026
vix = vix_level / 100
```

In [37]:
```
spy = df_all[(df_all['STOCK']=='SPY') & (~df_all['IV_final'].isna())].copy()
spy['days'] = spy['T'] * 365.0

spy_atm = spy[np.abs(spy['moneyness'] - 1.0) <= 0.02].copy()

target_maturity = spy_atm.iloc[(spy_atm['days'] - 30).abs().argsort()[:1]]['MA
spy_30d_atm_iv = spy_atm[spy_atm['MATURITY'] == target_maturity]['IV_final'].m

print("VIX :", vix)
print("SPY ATM ~30d maturity:", target_maturity)
print("SPY ATM ~30d avg IV:", spy_30d_atm_iv)
```

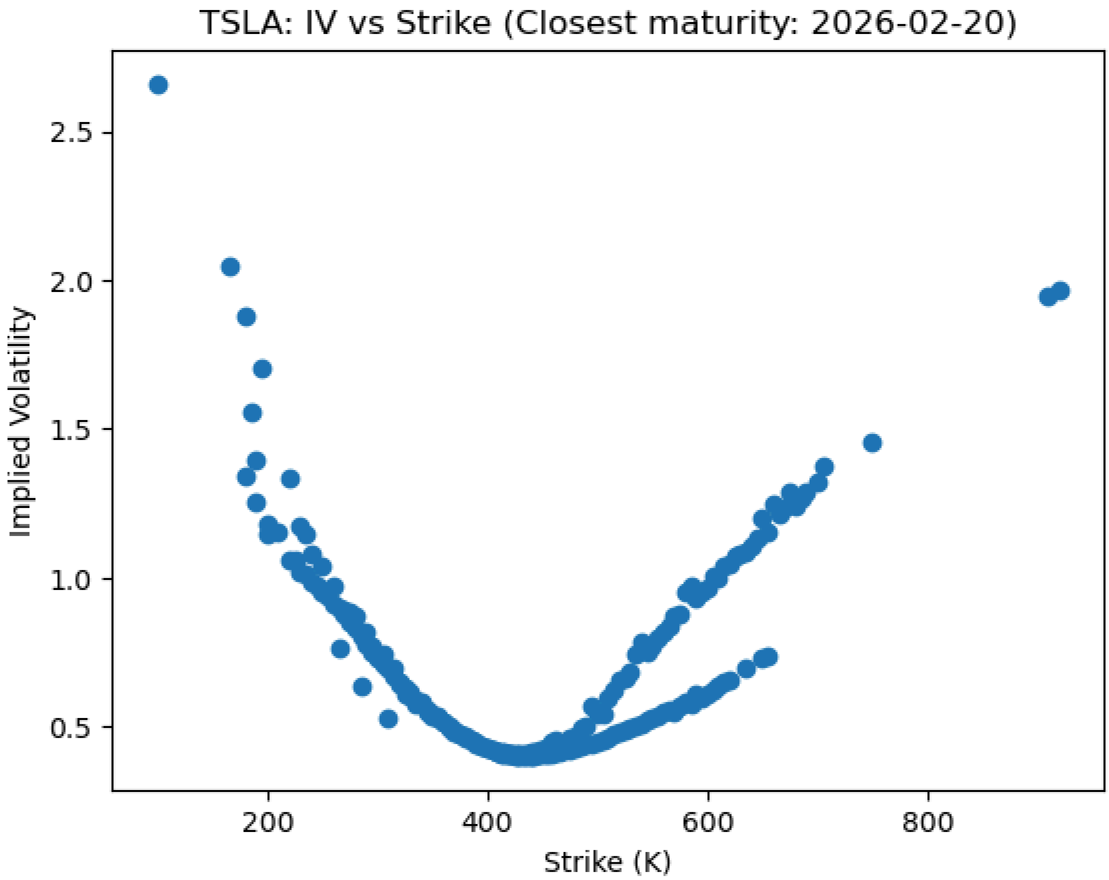
```
VIX : 0.1776
SPY ATM ~30d maturity: 2026-03-20
SPY ATM ~30d avg IV: 0.1546442644013419
```

Implied Volatility Plots

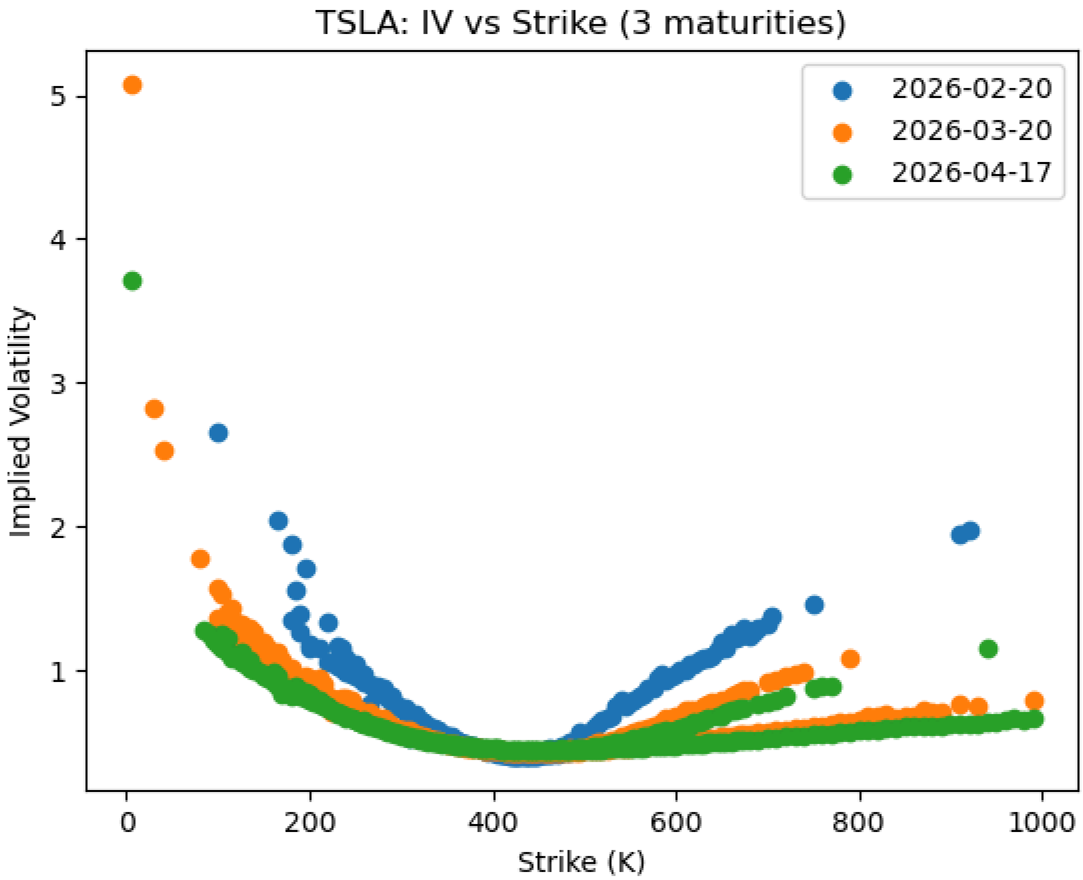
We visualize the implied volatilities in a few ways:

- **2D (Closest Maturity):**
  Plot implied volatility vs. strike (K) for the nearest expiration to observe the volatility smile or skew.
- **2D (Three Maturities):**
  Plot implied volatility vs. strike for all three maturities on the same graph, using different colors for each.
- **3D Plot (Bonus):**
  Create a 3D plot of implied volatility as a function of strike and maturity to visualize the volatility surface.

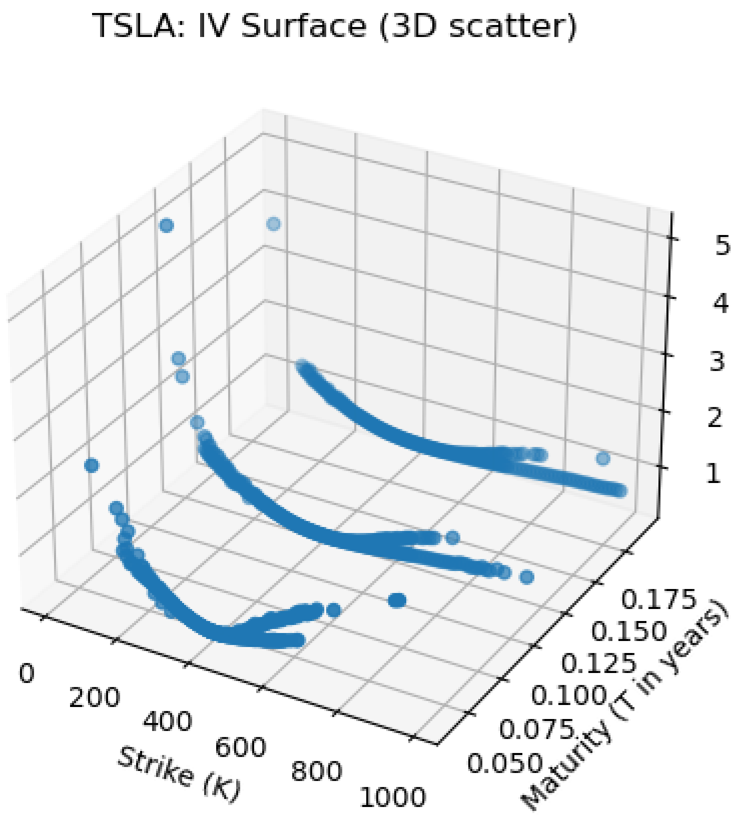
In [38]:
```
stock = 'TSLA'  # change to 'SPY'
d = df_all[(df_all['STOCK'] == stock) & (~df_all['IV_final'].isna())].copy()

# get the closest maturity date (smallest T)
closest_mat = d.loc[d['T'].idxmin(), 'MATURITY']
d_closest = d[d['MATURITY'] == closest_mat].copy()

plt.figure()
plt.scatter(d_closest['K'], d_closest['IV_final'])
plt.xlabel("Strike (K)")
plt.ylabel("Implied Volatility")
plt.title(f"{stock}: IV vs Strike (Closest maturity: {closest_mat})")
plt.show()
```



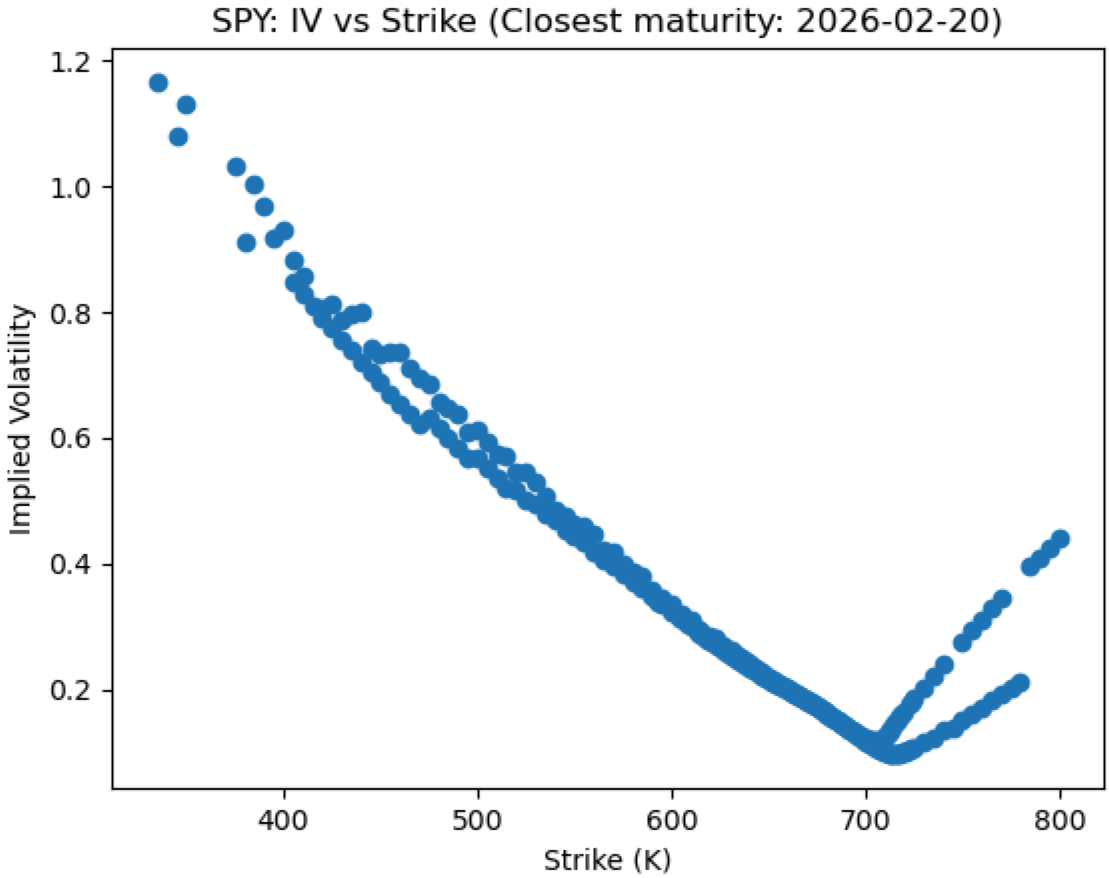
In [39]:
```
mats = sorted(d['MATURITY'].unique())[:3]
plt.figure()
for m in mats:
    dm = d[d['MATURITY'] == m]
    plt.scatter(dm['K'], dm['IV_final'], label=str(m))

plt.xlabel("Strike (K)")
plt.ylabel("Implied Volatility")
plt.title(f"{stock}: IV vs Strike (3 maturities)")
plt.legend()
plt.show()
```



In [40]:
```
from mpl_toolkits.mplot3d import Axes3D

d3 = d[d['MATURITY'].isin(mats)].copy()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(d3['K'], d3['T'], d3['IV_final'])

ax.set_xlabel("Strike (K)")
ax.set_ylabel("Maturity (T in years)")
ax.set_zlabel("Implied Volatility")
ax.set_title(f"{stock}: IV Surface (3D scatter)")
plt.show()
```

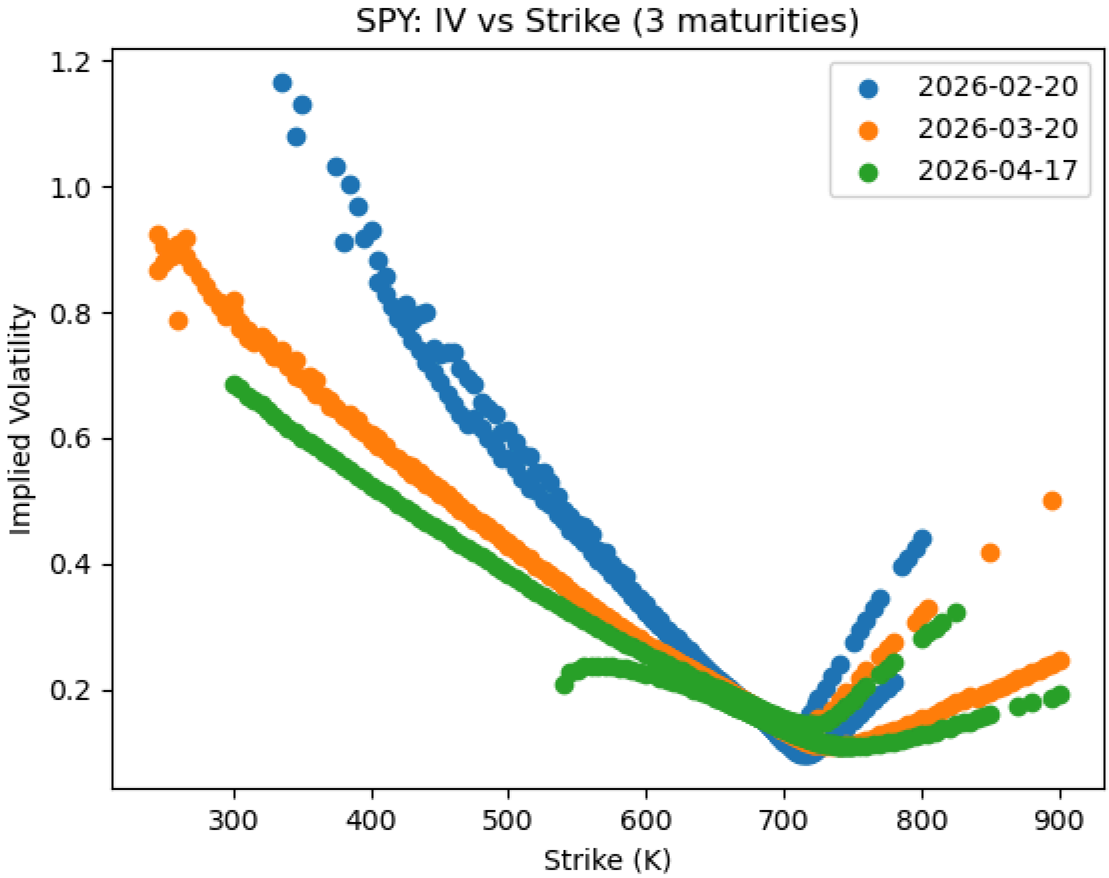


In [41]:
```
stock = 'SPY'  # change to 'SPY'
d = df_all[(df_all['STOCK'] == stock) & (~df_all['IV_final'].isna())].copy()

# get the closest maturity date (smallest T)
closest_mat = d.loc[d['T'].idxmin(), 'MATURITY']
d_closest = d[d['MATURITY'] == closest_mat].copy()

plt.figure()
plt.scatter(d_closest['K'], d_closest['IV_final'])
plt.xlabel("Strike (K)")
plt.ylabel("Implied Volatility")
plt.title(f"{stock}: IV vs Strike (Closest maturity: {closest_mat})")
plt.show()
```

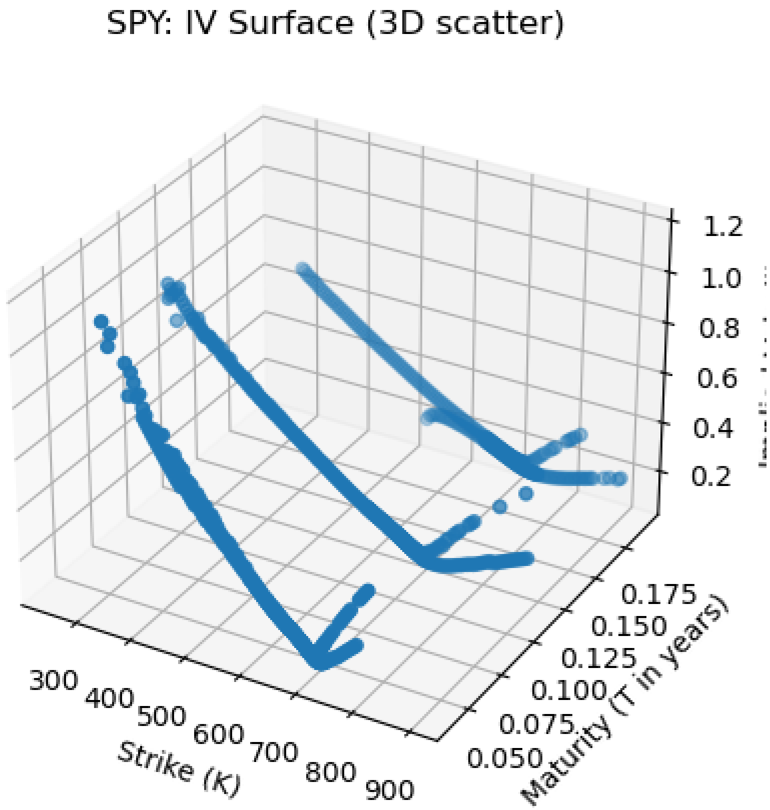


In [42]:

```
mats = sorted(d['MATURITY'].unique())[:3]
plt.figure()
for m in mats:
    dm = d[d['MATURITY'] == m]
    plt.scatter(dm['K'], dm['IV_final'], label=str(m))

plt.xlabel("Strike (K)")
plt.ylabel("Implied Volatility")
plt.title(f"{stock}: IV vs Strike (3 maturities)")
plt.legend()
plt.show()
```

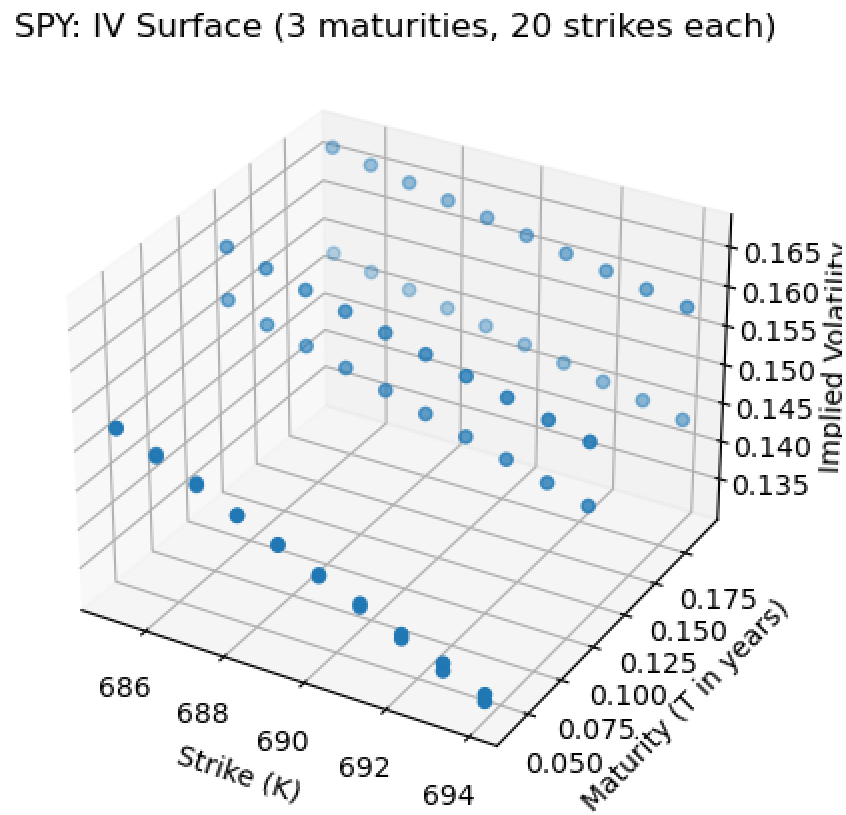


In [43]:
```
from mpl_toolkits.mplot3d import Axes3D

d3 = d[d['MATURITY'].isin(mats)].copy()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(d3['K'], d3['T'], d3['IV_final'])

ax.set_xlabel("Strike (K)")
ax.set_ylabel("Maturity (T in years)")
ax.set_zlabel("Implied Volatility")
ax.set_title(f"{stock}: IV Surface (3D scatter)")
plt.show()
```



In [44]:
```
S0 = float(d['S0'].iloc[0]) if 'S0' in d.columns else None

def pick_20_atm(dm, S0):
    dm = dm.copy()
    dm['dist'] = (dm['K'] - S0).abs()
    return dm.sort_values('dist').head(20)

d20 = []
for m in mats:
    dm = d[d['MATURITY'] == m]
    if S0 is not None:
        dm = pick_20_atm(dm, S0)
    else:
        dm = dm.sort_values('K').head(20)
    d20.append(dm)

d20 = pd.concat(d20, ignore_index=True)
```

In [45]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(d20['K'], d20['T'], d20['IV_final'])
ax.set_xlabel("Strike (K)")
ax.set_ylabel("Maturity (T in years)")
ax.set_zlabel("Implied Volatility")
ax.set_title(f"{stock}: IV Surface (3 maturities, 20 strikes each)")
plt.show()
```



Greeks

Finally, we compute the Greeks using the Black–Scholes formulas.

For each option, we calculate:

- **Delta** – Sensitivity to changes in the stock price
- **Vega** – Sensitivity to volatility
- **Gamma** – Sensitivity of Delta

These measures help us understand the risk profile of TSLA and SPY options and how their prices react to market changes.

In [46]:
```
g = df_all[(df_all['OPT_PUT_CALL'].str.lower() == 'call') & (df_all['T'] > 0)]
g = g.dropna(subset=['S0','K','T','IV_final'])

S = g['S0'].to_numpy(float)
K = g['K'].to_numpy(float)
T = g['T'].to_numpy(float)
sig = g['IV_final'].to_numpy(float)

sqrtT = np.sqrt(T)
d1 = (np.log(S / K) + (r + 0.5 * sig**2) * T) / (sig * sqrtT)

g['Delta_BS'] = norm.cdf(d1)
g['Vega_BS']  = S * norm.pdf(d1) * sqrtT
g['Gamma_BS'] = norm.pdf(d1) / (S * sig * sqrtT)
```

In [47]:
```
h = 0.01 * S
eps = 1e-4
```

In [48]:
```
def bs_call_price_vec(S, K, T, r, sigma):
    sqrtT = np.sqrt(T)
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * sqrtT)
    d2 = d1 - sigma * sqrtT
    return S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
```

In [49]:
```
C0  = bs_call_price_vec(S,     K, T, r, sig)
Cph = bs_call_price_vec(S + h, K, T, r, sig)
Cmh = bs_call_price_vec(S - h, K, T, r, sig)

g['Delta_FD'] = (Cph - Cmh) / (2 * h)
g['Gamma_FD'] = (Cph - 2*C0 + Cmh) / (h**2)

Cpe = bs_call_price_vec(S, K, T, r, sig + eps)
Cme = bs_call_price_vec(S, K, T, r, sig - eps)

g['Vega_FD']  = (Cpe - Cme) / (2 * eps)
```

In [50]:
```
g['Delta_diff'] = (g['Delta_BS'] - g['Delta_FD']).abs()
g['Vega_diff']  = (g['Vega_BS']  - g['Vega_FD']).abs()
g['Gamma_diff'] = (g['Gamma_BS'] - g['Gamma_FD']).abs()
```

In [51]:
```
greeks_table = g[['STOCK','MATURITY','K','T','IV_final','Delta_BS','Delta_FD',

greeks_table.head()
```

Out[51]:

	STOCK	MATURITY	K	T	IV_final	Delta_BS	Delta_FD	Delta_diff	Vega_BS
766	SPY	2026-02-20	335.0	0.038356	1.164771	0.999487	0.999485	0.000002	0.245741
767	SPY	2026-02-20	345.0	0.038356	1.077205	0.999654	0.999652	0.000002	0.170434
768	SPY	2026-02-20	350.0	0.038356	1.128136	0.999274	0.999272	0.000003	0.338383
769	SPY	2026-02-20	375.0	0.038356	1.032611	0.999090	0.999086	0.000004	0.416815
770	SPY	2026-02-20	380.0	0.038356	0.911694	0.999701	0.999699	0.000002	0.148982

Pricing Options Using DATA2

Now we move to DATA2.

For each strike price, we:

- Use the stock price from DATA2
- Use the implied volatility calculated from DATA1
- Use the short-term interest rate for the DATA2 date

Then, we plug these values into the Black–Scholes formula to compute the theoretical option price.

This allows us to see how well yesterday's implied volatility explains today's option prices.

In [52]:
```
BASE_2 = "/Users/simratkaurrandhawa/Desktop/Sem 4/fe621/Assignments/DATA2"

meta_2 = pd.read_csv(os.path.join(BASE_2, "META.csv"))
snap_2 = pd.read_csv(os.path.join(BASE_2, "SNAP_TSLA_SPY.csv"))

tsla_opt_2 = pd.read_csv(os.path.join(BASE_2, "TSLA_US_Equity_OPTIONS_NEXT3MONT
spy_opt_2  = pd.read_csv(os.path.join(BASE_2, "SPY_US_Equity_OPTIONS_NEXT3MONTI

asof_date_2 = datetime.strptime(meta_2.loc[0, "asof_date"], "%Y-%m-%d").date()

S0_tsla_2 = float(snap_2.loc[0, "TSLA.US.Equity.PX_LAST"])
S0_spy_2  = float(snap_2.loc[0, "SPY.US.Equity.PX_LAST"])
```

In [53]:
```
r_2 = 0.0364   # effective Fed funds rate around Feb 9
```

In [54]:
```
iv_lookup = df_all[['STOCK','MATURITY','K','OPT_PUT_CALL','IV_final']].copy()
iv_lookup['MATURITY'] = pd.to_datetime(iv_lookup['MATURITY']).dt.date
```

In [55]:
```
def prep_opt(df, asof_date):
    d = df[
        (df['VOLUME'].fillna(0) > 0)
    ].dropna(subset=['BID','ASK','OPT_STRIKE_PX','MATURITY','OPT_PUT_CALL']).c

    d['MID'] = 0.5 * (d['BID'] + d['ASK'])
    d['K'] = d['OPT_STRIKE_PX'].astype(float)

    d['MATURITY'] = pd.to_datetime(d['MATURITY'])
    d['T'] = (d['MATURITY'].dt.date - asof_date).dt.days / 365.0

    return d[d['T'] > 0]
```

In [56]:
```
df_tsla2 = prep_opt(tsla_opt_2, asof_date_2)
df_spy2  = prep_opt(spy_opt_2,  asof_date_2)
```

In [57]:
```
df_tsla2['MATURITY'] = pd.to_datetime(df_tsla2['MATURITY']).dt.date
iv_lookup['MATURITY'] = pd.to_datetime(iv_lookup['MATURITY']).dt.date
```

In [58]:
```
df_tsla2['STOCK'] = 'TSLA'
df_tsla2 = df_tsla2.merge(iv_lookup, on=['STOCK','MATURITY','K','OPT_PUT_CALL']
```

In [59]:
```
df_tsla2['BS_PRICE'] = df_tsla2.apply(
    lambda row: black_sholes(S0_tsla_2, row['K'], row['T'], r_2, row['IV_final
    if pd.notna(row['IV_final']) else np.nan,
    axis=1
)

df_tsla2['BS_minus_MID'] = df_tsla2['BS_PRICE'] - df_tsla2['MID']
```

In [60]:
```
print("Total rows:", len(df_tsla2))
print("Priced rows:", df_tsla2['BS_PRICE'].notna().sum())
```

```
print("missing BS_PRICE:", df_tsla2['BS_PRICE'].isna().mean())
```

```
Total rows: 763
Priced rows: 706
missing BS_PRICE: 0.07470511140235911
```

In [61]:
```
df_tsla2['BS_minus_MID'].describe()
```

Out[61]:
```
count    706.000000
mean       0.483739
std        2.505447
min       -4.221311
25%       -0.097146
50%        0.141837
75%        3.251512
max        5.383621
Name: BS_minus_MID, dtype: float64
```

In [62]:
```
df_tsla2.groupby('MATURITY')['BS_minus_MID'].describe()
```

Out[62]:

MATURITY	count	mean	std	min	25%	50%	75%	max
2026-02-20	188.0	0.945885	2.730663	-3.991107	-0.039681	0.740152	3.431991	5.383621
2026-03-20	249.0	0.206476	2.438724	-4.221311	-0.609584	0.068261	1.622399	4.401005
2026-04-17	269.0	0.417402	2.362388	-4.091053	-0.080465	0.163855	1.562036	4.646851

Summary of Numerical Results

Table 1 shows the comparison between implied volatility estimates obtained using the bisection and Newton methods.

Table 2 summarizes the absolute deviations from put–call parity.

Table 3 reports the mean absolute errors between analytical and finite-difference Greeks.

Table 4 presents the absolute pricing errors when applying DATA1 implied volatility to price DATA2 options.

In [64]:
```
iv_comparison = pd.DataFrame({"Max |IV_bisect - IV_newton|": [(df_all["IV_bise

iv_comparison
```

Out[64]:

	Max |IV_bisect - IV_newton|	Mean |IV_bisect - IV_newton|
0	5.851142e-07	1.787404e-07

In [65]:
```
parity_table = df_all["abs_diff"].describe()[["mean", "50%", "75%", "max"]]
parity_table = parity_table.to_frame(name="abs_diff")

parity_table
```

Out[65]:

	abs_diff
mean	1.787404e-07
50%	1.249284e-07
75%	3.271623e-07
max	5.851142e-07

In [66]:

```
greeks_table = pd.DataFrame({"Greek": ["Delta", "Gamma", "Vega"],"Mean Absolute

greeks_table
```

Out[66]:

	Greek	Mean Absolute Error
0	Delta	0.000641
1	Gamma	0.000017
2	Vega	0.000002

In [67]:

```
pricing_error_table = df_tsla2["BS_minus_MID"].abs().describe()[["mean", "50%"
pricing_error_table = pricing_error_table.to_frame(name="|BS - MID|")

pricing_error_table
```

Out[67]:

	|BS - MID|
mean	1.919978
50%	1.716107
75%	3.531711
max	5.383621

In [ ]: