Homework 1

FE621 Computational Finance

due 23:59, Sunday February 15, 2026

For all the problems in this assignment you need to design and use a computer program, output results and present the results in nicely formatted tables and figures. The computer program may be written in any programming language you want. Please write comments to all the parts of your code. They are a requirement and they will be graded.

You need to submit a PDF containing the report. Please use a word processor such as Microsoft Word, LaTeX, or whatever Apple uses to create your report. You will be judged by the quality of the writing and the interpretation of the results.

Part 1. (20 points) Data gathering component

1. Write a function (program) to connect to sources and download data from one of the following sources:

   (a) GOOGLE finance http://www.google.com/finance

   (b) Yahoo Finance http://finance.yahoo.com

   (c) Bloomberg

   Notes. For extra credit you can turn in code to download data from the other two sources. Please note that the program needs to download both option data and equity data. For this problem (and only for this problem) you may use any built in function or toolbox that will facilitate your work. The data will have to be clean (no duplicated values, only one exchange, every column labeled properly, in other words, consolidated).

Note on Bloomberg data. For the Bloomberg source, access to one of Bloomberg terminals in the lab is required. If you use Bloomberg data, you may use the API to download data in Excel automatically and organize the data. However, this should be accomplished with an automatic script. If you use R to interface with the Bloomberg data, a useful package for that is Rblpapi, but there are other packages. For the online students, who do not have access to Bloomberg terminals, you may read about the package quantmod in R to download yahoo and google data automatically.

Bonus (5 points) Create a program that is capable of downloading multiple assets, combine them with the associated time column, and save the data into a csv or excel file.

2. With the function created in problem 1, download data on options and equity for the following symbols:

   - TSLA
   - SPY
   - ^VIX

   for two consecutive days (does not matter when, but no later than February 14^{th}) during the trading day (9:30am to 4:00pm). Please record the asset values (both TSLA and SPY) at the time when downloading is done. Please do the same with the ^VIX. For the options please note that the traditional options are maturing third Friday of each month. However note there are a lot more options available to download. Please give your quant student explanation why there are so many maturities available. For the assignment please download option chain data for the next three months (options expiring on the third Friday of the month).

   We shall refer to the data sets gathered in the two consecutive days as DATA1 (for the first day) and DATA2 (for the second day) respectively throughout this assignment and the following ones.

3. Write a paragraph describing the symbols you are downloading data for. Explain what is SPY and its purpose. (Hint: look up the definition of an ETF). Explain what is ^VIX and its purpose. Understand the

options' symbols. Understand when each option expires. Write this information and turn it in.

4. The following items will also need to be recorded:

   - The underlying equity, ETF, or index price at the exact moment when the rest of the data is downloaded.
   - The short-term interest rate which may be obtained here: http://www.federalreserve.gov/releases/H15/Current/. There are a lot of rates posted on the site - they are all yearly, *they are in percents and need to be converted to numbers.* There is no theoretical recommendation on which to use, I used to use 3-months Treasury bills which are not available anymore. Since then I have been using the "Federal funds (effective)" rate but you can go ahead and try others. You should remember to be consistent in your choice. Also, make sure that the interest rate that you use is for the same day when the data you use for the implied volatility was gathered and note that the data is typically quoted in percents (you will need numbers). The same site has a link to past (historical) interest rates.
   - Time to Maturity.

Part 2. (50 points) Analysis of the data.

5. Using your choice of computer programming language implement the Black-Scholes formulas as a function of current stock price S_0, volatility σ, time to expiration $T - t$ (in years), strike price K and short-term interest rate r (annual). Please note that no toolbox function is allowed but you may call the normal CDF function (e.g., `pnorm` in `R` or `scipy.stats.norm.cdf` in `Python`).

6. Implement the Bisection method to find the root of arbitrary functions. Apply this method to calculate the implied volatility on the first day you downloaded (DATA1). For this purpose use as the option value the average of bid and ask price if they both exist (and if their corresponding volume is nonzero). Also use a tolerance level of 10^{-6}. Report the implied volatility at the money (for the option with strike price closest to the traded stock price). You need to do it for both the stock and the

ETF data you have (you do not need to do this for ^VIX). Then average all the implied volatilities for the options between in-the-money and out-of-the-money.
Note. There is no clearly defined boundary between options at-the-money and out-of-the-money or in-the-money options. If we define moneyness as the ratio between S_0 the stock price today and K the strike price of the option some people use values of moneyness between 0.95 and 1.05 to define the options at the money. Yet, other authors use between 0.9 and 1.1. Use these guidelines if you wish to determine which options' implied volatilities should be averaged.

7. Implement the Newton method/Secant method or Muller method to find the root of arbitrary functions. You will need to discover the formula for the option's derivative with respect to the volatility σ. Apply these methods to the same options as in the previous problem. Compare the time it takes to get the root with the same level of accuracy.

8. Present a table reporting the implied volatility values obtained for every maturity, option type and stock. Also compile the average volatilities as described in the previous point. Comment on the observed difference in values obtained for TSLA and SPY. Compare with the current value of the ^VIX. Comment on what happens when the maturity increases. Comment on what happen when the options become in the money respectively out of the money.

9. For each option in your table calculate the price of the different type (Call/Put) using the Put-Call parity (please see Section 4 from [2]). Compare the resulting values with the BID/ASK values for the corresponding option if they exist.

10. Consider the implied volatility values obtained in the previous parts. Create a 2 dimensional plot of implied volatilities versus strike K for the closest to maturity options. What do you observe? Plot all implied volatilities for the three different maturities on the same plot, where you use a different color for each maturity. In total there should be 3 sets of points plotted with different color. (**Bonus** 5 Points) Create a 3D plot of the same implied vols as a function of both maturity and strike, i.e.: $\sigma(\tau_i, K_j)$ where $i = 1, 2, 3$, and $j = 1, 2, \ldots, 20$.

11. (Greeks) Calculate the derivatives of the call option price with respect to S (Delta), and σ (Vega) and the second derivative with respect to S (Gamma). First use the Black Scholes formula then approximate these derivatives using an approximation of the partial derivatives. Compare the numbers obtained by the two methods. Output a table containing all derivatives thus calculated.

12. Next we will use the second dataset DATA2. For each strike price in the data use the Stock price for the same day, the implied volatility you calculated from DATA1 and the current short-term interest rate (corresponding to the day on which DATA2 was gathered). Use the Black-Scholes formula, to calculate the option price.

Part 3. (30 points) Numerical Integration of real-valued functions. AMM Arbitrage Fee Revenue

AMMs are decentralized exchanges that quote prices using pool reserves rather than an order book. Compared to Traditional Finance order books, a key advantage is **continuous liquidity from the pool without needing a matching counterparty**. Liquidity Providers (LPs) earn revenue mainly from **swap fees**, so selecting a good fee rate γ under different volatility levels matters [5].

For simplicity we consider the following Constant Product Market Maker (CPMM) for a **BTC/USDC** pool. These are the pool elements:

- x_t: BTC reserves at time t
- y_t: USDC reserves at time t
- pool mid price is calculated as $P_t = \dfrac{y_t}{x_t}$ (USDC per BTC)
- external market price S_t (USDC per BTC)
- fee rate $\gamma \in (0, 1)$

The pool satisfies the constant-product rule, that is at every moment in time the product of quantities must stay constant.

$$x_{t+1}y_{t+1} = x_t y_t = k.$$

Next we describe the trade mechanics. Suppose someone wants to trade Δx with the pool. That results in a Δy obtained from the pool. This is done so that the updated reserves continue to satisfy the constant product rule. Specifically, using the γ rate that is assesed we must have:

$$(x_t + (1-\gamma)\Delta x)(y_t - \Delta y) = k$$

This will satisfy the constraint $x_{t+1}y_{t+1} = x_t y_t = k$. Note that the price ratio of the assets is changing after this trade: $P_{t+1} = \dfrac{y_t - \Delta y}{x_t + (1-\gamma)\Delta x}$.

Outside this mechanics as long as the quantity of assets in the pool stay constant the price does not change. We are considering a situation where the outside price S_t moves. If S_{t+1} leaves the no-arbitrage band, $\left[P_t(1-\gamma), \frac{P_t}{1-\gamma}\right]$ then an arbitrage opportunity arises. We assume that arbitragers act optimally and instantly. As a result the following happens.

Case 1: $S_{t+1} > P_t \dfrac{1}{1-\gamma}$ **(BTC cheaper in the pool)**

Arbitragers will swap USDC $\to$ BTC until the resulting band after the trades moves so that it contains the outside price S_{t+1}. Updates:

$$x_{t+1} = x_t - \Delta x, \qquad y_{t+1} = y_t + (1-\gamma)\Delta y, \qquad \Delta x, \Delta y > 0,$$

with boundary condition

$$P_{t+1}\frac{1}{1-\gamma} = \frac{y_{t+1}}{x_{t+1}}\frac{1}{1-\gamma} = S_{t+1},$$

The corresponding fee revenue (USDC) is coming from the input asset y. It is thus equal to $\gamma\,\Delta y$.

Case 2: $S_{t+1} < P_t(1-\gamma)$ **(BTC cheaper outside)**

Arbitragers will swap BTC $\to$ USDC and the updated reserves are:

$$x_{t+1} = x_t + (1-\gamma)\Delta x, \qquad y_{t+1} = y_t - \Delta y, \qquad \Delta x, \Delta y > 0,$$

with boundary condition

$$P_{t+1}(1-\gamma) = \frac{y_{t+1}}{x_{t+1}}(1-\gamma) = S_{t+1},$$

and the fee revenue is $\gamma\Delta x$. After converting BTC fee to USDC using S_{t+1} is calculated as $\gamma\,\Delta x\,S_{t+1}$.

Questions:

(a) (10 pts) Derive the swap amounts

Under both Case 1 and Case 2, use the corresponding boundary condition, and the reserve updates above, to derive the swap size:

$$\Delta x(S_{t+1}; x_t, y_t, \gamma, k), \quad \Delta y(S_{t+1}; x_t, y_t, \gamma, k)$$

such that the one-step fee revenue is:

$$R(S_{t+1}) = \mathbf{1}_{\left\{S_{t+1} > \frac{P_t}{1-\gamma}\right\}} \gamma \, \Delta y + \mathbf{1}_{\{S_{t+1} < P_t(1-\gamma)\}} \gamma \, \Delta x \, S_{t+1}.$$

where we used the notation $1_{\{.\}}$ for the indicator function.

(b) (10 pts) Expected fee revenue

Assume the initial BTC/USDC pool reserves are $x_t = y_t = 1000$, so $P_t = \frac{y_t}{x_t} = 1$. Let $S_t = 1$ and $\Delta t = \frac{1}{365}$. Assume the external price follows one-step GBM:

$$S_{t+1} = S_t \exp\Big(-\tfrac{1}{2}\sigma^2 \Delta t + \sigma\sqrt{\Delta t}\, Z\Big), \qquad Z \sim N(0,1),$$

so S_{t+1} is lognormal with density $f_{S_{t+1}}(s)$.

Using the setup above and the solution from (a), the expected one-step fee revenue is:

$$\mathbb{E}[R(S_{t+1})] = \int_{P_t/(1-\gamma)}^{\infty} \gamma \Delta y(S_{t+1}; x_t, y_t, \gamma, k)\, f_{S_{t+1}}(s)\, ds + \int_0^{P_t(1-\gamma)} \gamma \Delta x(S_{t+1}; x_t, y_t, \gamma, k) S_{t+1}\, f_{S_{t+1}}(s)\, ds.$$

Task: Numerically approximate $\mathbb{E}[R(S_{t+1})]$ using trapezoidal rule learned in class.

(c) (10 pts) Optimal Fee Rate under different volatilities

Use $\sigma \in \{0.2, 0.6, 1.0\}$ and $\gamma \in \{0.001, 0.003, 0.01\}$. For each σ, compute $\mathbb{E}[R]$ for the three fee rates and select:

$$\gamma^*(\sigma) = \arg\max_{\gamma} \mathbb{E}[R(S_{t+1})].$$

Construct a table reporting the $\mathbb{E}[R]$ values and the best $\gamma^*(\sigma)$ among the 3 options.

Then for each $\sigma \in [0.1, 1.0]$ on a grid (e.g. 0.01 step), compute the optimal $\gamma^*(\sigma)$. Finally, produce a scatter plot or line plot for σ vs. $\gamma^*(\sigma)$, and comment briefly on the pattern you observe.

Part 4. (**Bonus** 10 points) Consider the following functions:

$$f_1(x, y) = xy$$
$$f_2(x, y) = e^{x+y}$$

1. Analytically solve the following integral for both f_1 and f_2

$$\int_0^1 \int_0^3 f_i(x, y) \mathrm{d}y \mathrm{d}x$$

2. Calculate the numerical integral of the f_1 and f_2 by applying the trapezoidal rule for double integral as discussed in [3], Page 118-119. Please choose four different pairs of values for $(\Delta x, \Delta y)$. Use these values to approximate the double integral for both f_1 and f_2. Calculate the error of the approximation for each choice of values. Comment on the results.

   Hint 1 First, discretize the x domain into n+1 points Δx apart, where $x_0 = 0$ and $x_{n+1} = 1$, and the y domain into $m + 1$ points Δy apart, where $y_0 = 0$ and $y_{m+1} = 3$. The composite trapezoidal rule approximates the integral as

$$\int_0^1 \int_0^3 f(x, y) \mathrm{d}y \mathrm{d}x \approx \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\Delta x \Delta y}{16} [f(x_i, y_j) + f(x_i, y_{j+1}) + f(x_{i+1}, y_j)$$
$$+ f(x_{i+1}, y_{j+1}) + 2 \left(f(\frac{x_i + x_{i+1}}{2}, y_j) + f(\frac{x_i + x_{i+1}}{2}, y_{j+1}) + f(x_i, \frac{y_j + y_{j+1}}{2}) \right.$$
$$\left. + f(x_{j+1}, \frac{y_j + y_{j+1}}{2}) \right) + 4 f(\frac{x_i + x_{i+1}}{2}, \frac{y_j + y_{j+1}}{2})]$$

   Hint 2. Please note the numbers chosen for $(\Delta x, \Delta y)$ should be specific to each student in the class.

References

[1] Clewlow, Les and Strickland, Chris. *Implementing Derivative Models (Wiley Series in Financial Engineering)*, John Wiley & Sons 1996.

[2] Mariani, Maria C. and Florescu, Ionut *Quantitative Finance*, 2020, John Wiley & Sons.

[3] Rouah, F. D. *The Heston Model and Its Extensions in Matlab and C*, 2013, John Wiley & Sons.

[4] Mikhailov, Sergei and Nögel, Ulrich. "Heston's stochastic volatility model: Implementation, calibration and some extensions" *Wilmott Journal*, 2004.

[5] Angeris, Guillermo and Kao, Hsien-Tang and Chiang, Rei and Noyes, Charlie and Chitra, Tarun. "An analysis of Uniswap markets" *arXiv preprint arXiv:1911.03380*, 2019.

Part 1: Data gathering component

In [2]:
```
import os
import yfinance as yf # https://ranaroussi.github.io/yfinance/
import pandas as pd
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

1.

In [3]:
```
# download asset(s) data, time-indexed and save to file
def get_ohlcv(tickers, period='1d', interval='1m', start=None, end=None, save=True):

    if isinstance(tickers, str):
        tickers = [tickers]

    # use yfinance download
    df = yf.download(tickers=tickers,
                     period=None if (start or end) else period,
                     interval=interval,
                     start=start,
                     end=end,
                     auto_adjust=True,
                     progress=False
                    )

    # save to file
    if save:
        filepath = f'{os.getcwd()}/ohlcv_ALL_{str(df.index[0]).split()[0]}({interval}).csv'
        df.to_csv(filepath)
        print(f'Saved: {filepath}')

    return df

# download option(s) data, time-indexed and save to file
def get_options(tickers, exp_months=3, save=True):

    if isinstance(tickers, str):
        tickers = [tickers]

    # check if traditional third friday of each month
    def valid_friday(ts):
        d = pd.Timestamp(ts)
        return d.weekday() == 4 and (15 <= d.day <= 21)  # 3rd Friday

    rows = []
    for tick in tickers:
        t = yf.Ticker(tick)

        # only get options that expire on a traditional third friday
        expiries = [e for e in t.options if valid_friday(e)][:exp_months]

        # spot price right now
        temp = yf.download(tick, period='1d', interval='1m', auto_adjust=True, progress=False)
        S0 = temp['Close'].iloc[-1]
        S0_time = temp.index[-1]

        for exp in expiries:
            chain = t.option_chain(exp)
            calls = chain.calls.copy()
            puts  = chain.puts.copy()

            calls['type'] = 'call'
```

```
            puts['type']  = 'put'

            calls['ticker'] = tick
            puts['ticker'] = tick

            calls['expiry'] = exp
            puts['expiry'] = exp

            # save spot price as well for (Q4.)
            calls['S0'] = float(S0)
            puts['S0'] = float(S0)
            calls['S0_time'] = S0_time.date()
            puts['S0_time'] = S0_time.date()

            # combine calls and puts
            df = pd.concat([calls, puts], ignore_index=True)
            rows.append(df)

    options_df = pd.concat(rows, ignore_index=True)

    # save to file
    if save:
        filepath = f'{os.getcwd()}/options_{S0_time.strftime("%Y-%m-%d_%H%M")}.csv'
        options_df.to_csv(filepath, index=False)
        print(f'Saved: {filepath}')

    return options_df
```

2.

Historical OHLCV Data

In [4]:
```
# not saving bc I already have the ones I wanted saved.
get_ohlcv(tickers=['TSLA','SPY','^VIX'], start='2026-02-04', end='2026-02-05', interval='1m', save=False);
```

Option Data

In [5]:
```
# not saving bc I already have the ones I wanted saved.
options = get_options(tickers=['TSLA','SPY','^VIX'], exp_months=3, save=False)
```

In [6]:
```
data1 = pd.read_csv('saved_options_2026-02-04_1834.csv')
data2 = pd.read_csv('saved_options_2026-02-05_1754.csv')
```

3.

TSLA: This is the ticker for Tesla. Tesla is an American electric car company that specializes in EV's but also produces solar panels, battery storage, and automated driving.

SPY: This is the ticker for the S&P 500 exchange-traded fund (ETF). SPY tracks the performance of the S&P 500 index. SPY is generally used to track the performance of the US economy because it covers 500 of the largest companies across various industries in the US economy.

^VIX: This is the ticker for the CBOE volatility index. VIX measures the market's expected volatility over the next 30 days. It is often called the "fear index" because high readings correlate with market uncertainty and high volatility.

After looking at the option chain above for 'TSLA' with expiry '2026-03-20', the option symboling scheme refers to the asset, expiration, call/put, and strike price. In my case, the option dictionary is made to where all options will be shown for the key (ASSET, EXPIRATION). So above, the options under ('TSLA', '2026-03-20') will all expire on '2026-03-20'.

4.

In [7]:
```
# https://www.federalreserve.gov/releases/h15/
# using 'Federal funds (effective)' rate
FED_FUNDS_RATE = 0.0364

# finds T using business days (252)
def get_maturity(expire, today=str(pd.Timestamp.now().date())):
    # count business days
    days = np.busday_count(today, expire)

    # calc T
    T = days / 252.0

    return T
```

Part 2: Analysis of the data

5.

In [8]:
```
def bs_formula(cp, S0, vol, T, K, r):
    # calc d1/d2
    d1 = (np.log(S0 / K) + (r + vol**2 / 2) * T) / (vol * T**0.5)
    d2 = d1 - vol * T**0.5

    # return value depending on c (call), p (put)
    if cp.upper() == 'C':
        return S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    elif cp.upper() == 'P':
        return K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)
    else:
        print(f'unknown cp: {cp}')
        return -1
```

6.

In [9]:
```
def bisection(func, a, b, tol=1e-6):
    fa = func(a)
    fb = func(b)

    if fa * fb > 0:
        print(f'Check f(a) * f(b) < 0')
        return

    for i in range(10000):
        mid = (a + b) / 2.0
        fm = func(mid)

        # return if tol met
        if abs(b - a) < tol:
            return mid

        if fa * fm < 0:
            b = mid
            fb = fm
        elif fb * fm < 0:
            a = mid
            fa = fm
        else:
            return mid

    return (a + b) / 2.0

def calc_iv_bisection_atm(options_df):
    res = {}
```

```
    # iterate across all options with the same ticker and expiry
    for (ticker, exp), df in options_df.groupby(['ticker', 'expiry']):
        calls = df[df['type'] == 'call'].copy()

        S0 = calls['S0'].iloc[0]
        now = calls['S0_time'].iloc[0]
        T = get_maturity(exp, now)
        r = FED_FUNDS_RATE

        # get the ind of the option with the strike closest to S0
        atm_ind = (calls['strike'] - S0).abs().values.argmin()
        row = calls.iloc[atm_ind]

        K = row['strike']
        Cb = row['bid']
        Ca = row['ask']

        if Cb == 0 or Ca == 0:
            continue

        # creates lambda func to calc the root of calculated call - call price (avg bid ask)
        bs_func = lambda input_vol: bs_formula(cp='C', S0=S0, vol=input_vol, T=T, K=K, r=r) - ((Cb + Ca) / 2

        # use bisection to find the root
        bis_res = bisection(bs_func, 1e-6, 5.0)
        if bis_res:
            res[(ticker, exp)] = bis_res

    return res

def calc_iv_bisection_range(options_df, lo=0.9, hi=1.1):
    res = {}

    # iterate across all options with the same ticker and expiry
    for (ticker, exp), df in options_df.groupby(['ticker', 'expiry']):
        calls = df[df['type'] == 'call'].copy()

        S0 = calls['S0'].iloc[0]
        now = calls['S0_time'].iloc[0]
        T = get_maturity(exp, now)
        r = FED_FUNDS_RATE

        # parallel compute moneyness for all calls
        calls['moneyness'] = S0 / calls['strike']

        # get window of options in range
        window = calls[(calls['moneyness'] >= lo)
                       & (calls['moneyness'] <= hi)].copy()

        # calc each options' implied volatility
        imp_vols = []
        for _, row in window.iterrows():
            K = row['strike']
            Cb = row['bid']
            Ca = row['ask']

            if Cb == 0 or Ca == 0:
                continue

            # creates lambda func to calc the root of calculated call - call price (avg bid ask)
            bs_func = lambda input_vol: bs_formula(cp='C', S0=S0, vol=input_vol, T=T, K=K, r=r) - ((Cb + Ca)

            # use bisection to find the root
            bis_res = bisection(bs_func, 1e-6, 5.0)
            if bis_res:
                imp_vols.append(bis_res)

        res[(ticker, exp)] = np.mean(imp_vols)

    return res
```

At the Money

Below shows the implied volatilities of the option closest to at the money. The key refers to (asset, expiration).

In [10]:
```
%%time
iv_bisection_atm = calc_iv_bisection_atm(data1); iv_bisection_atm
```

```
CPU times: total: 31.2 ms
Wall time: 24 ms
```

Out[10]:
```
{('SPY', '2026-02-20'): 0.16067691076260804,
 ('SPY', '2026-03-20'): 0.15673287220662835,
 ('SPY', '2026-04-17'): 0.15099175397008652,
 ('TSLA', '2026-02-20'): 0.4604983468081356,
 ('TSLA', '2026-03-20'): 0.45203746917444476,
 ('TSLA', '2026-04-17'): 0.45527698097008473}
```

Moneyness (0.9 - 1.1)

Below shows the avg implied volatilities of options in the 'moneyness' range of 0.9 - 1.1. The key refers to (asset, expiration).

In [11]:
```
%%time
iv_bisection_range = calc_iv_bisection_range(data1, 0.9, 1.1); iv_bisection_range
```

```
Check f(a) * f(b) < 0
Check f(a) * f(b) < 0
CPU times: total: 766 ms
Wall time: 762 ms
```

Out[11]:
```
{('SPY', '2026-02-20'): 0.15840900759553495,
 ('SPY', '2026-03-20'): 0.15266280140755525,
 ('SPY', '2026-04-17'): 0.14010459963295743,
 ('TSLA', '2026-02-20'): 0.4588356259041329,
 ('TSLA', '2026-03-20'): 0.45471207806220654,
 ('TSLA', '2026-04-17'): 0.4581762620708123}
```

7.

Newton's Method

Wiki.

To use Newton's method with the BS formulas, we also need a function to return the derivative (slope) with input volatility. In other words, we need to find the BS formula's sensitivity to volatility which is Vega.

Newtons:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Vega:

$$\frac{\partial V}{\partial \sigma} = S_0\sqrt{T-t}N'(d_1)$$

In [12]:
```
def newton(func, d_func, x0, tol=1e-6, mute=True):
    x = x0
    for _ in range(10000):
        fx = func(x)
        if abs(fx) < tol:
            return x

        dfx = d_func(x)
```

```
        # constraint dfx != 0
        if dfx == 0:
            if not mute:
                print(f'Check d_func({x}) != 0')
            return

        x_next = x - fx / dfx
        # constraint x_n+1 > 0
        if x_next <= 0:
            if not mute:
                print(f'Check x_next: {x_next} > 0.')
            return

        x = x_next

    return x

def bs_vega(S0, vol, T, K, r):
    d1 = (np.log(S0 / K) + (r + vol**2 / 2) * T) / (vol * T**0.5)

    return S0 * norm.pdf(d1) * T**0.5

def calc_iv_newton_atm(options_df):
    res = {}
    Ks = []

    # iterate across all options with the same ticker and expiry
    for (ticker, exp), df in options_df.groupby(['ticker', 'expiry']):
        calls = df[df['type'] == 'call'].copy()

        S0 = calls['S0'].iloc[0]
        now = calls['S0_time'].iloc[0]
        T = get_maturity(exp, now)
        r = FED_FUNDS_RATE

        # get the ind of the option with the strike closest to S0
        atm_ind = (calls['strike'] - S0).abs().values.argmin()
        row = calls.iloc[atm_ind]

        K = row['strike']
        Cb = row['bid']
        Ca = row['ask']

        if Cb == 0 or Ca == 0:
            continue

        # creates lambda func to calc the root of calculated call - call price (avg bid ask)
        bs_func = lambda input_vol: bs_formula(cp='C', S0=S0, vol=input_vol, T=T, K=K, r=r) - ((Cb + Ca) / 2
        bs_vega_func = lambda input_vol: bs_vega(S0=S0, vol=input_vol, T=T, K=K, r=r)

        # use bisection to find the root
        newt_res = newton(bs_func, bs_vega_func, 0.3)
        if newt_res:
            res[(ticker, exp)] = newt_res
            Ks.append(K)
    return res, Ks

def calc_iv_newton_range(options_df, lo=0.9, hi=1.1):
    res = {}
    Ks = []
    ivs = []

    # iterate across all options with the same ticker and expiry
    for (ticker, exp), df in options_df.groupby(['ticker', 'expiry']):
        calls = df[df['type'] == 'call'].copy()

        S0 = calls['S0'].iloc[0]
        now = calls['S0_time'].iloc[0]
        T = get_maturity(exp, now)
        r = FED_FUNDS_RATE
```

```
        # parallel compute moneyness for all calls
        calls['moneyness'] = S0 / calls['strike']

        # get window of options in range
        window = calls[(calls['moneyness'] >= lo)
                       & (calls['moneyness'] <= hi)].copy()

        # calc each options' implied volatility
        imp_vols = []
        r_Ks = []
        for _, row in window.iterrows():
            K = row['strike']
            Cb = row['bid']
            Ca = row['ask']

            if Cb == 0 or Ca == 0:
                continue

            # creates lambda func to calc the root of calculated call - call price (avg bid ask)
            bs_func = lambda input_vol: bs_formula(cp='C', S0=S0, vol=input_vol, T=T, K=K, r=r) - ((Cb + Ca)
            bs_vega_func = lambda input_vol: bs_vega(S0=S0, vol=input_vol, T=T, K=K, r=r)

            # use bisection to find the root
            newt_res = newton(bs_func, bs_vega_func, 0.3)
            if newt_res:
                imp_vols.append(newt_res)
                r_Ks.append(K)

        Ks.append(r_Ks)
        ivs.append(imp_vols)
        res[(ticker, exp)] = np.mean(imp_vols)

    return res, Ks, ivs

def calc_ivs_newton(options_df):
    k_ivs = {}

    # iterate across all options with the same ticker and expiry
    for (ticker, exp), df in options_df.groupby(['ticker', 'expiry']):
        calls = df[df['type'] == 'call'].copy()

        S0 = calls['S0'].iloc[0]
        now = calls['S0_time'].iloc[0]
        T = get_maturity(exp, now)
        r = FED_FUNDS_RATE

        # calc each options' implied volatility
        for _, row in calls.iterrows():
            K = row['strike']
            Cb = row['bid']
            Ca = row['ask']

            if Cb == 0 or Ca == 0:
                continue

            # creates lambda func to calc the root of calculated call - call price (avg bid ask)
            bs_func = lambda input_vol: bs_formula(cp='C', S0=S0, vol=input_vol, T=T, K=K, r=r) - ((Cb + Ca)
            bs_vega_func = lambda input_vol: bs_vega(S0=S0, vol=input_vol, T=T, K=K, r=r)

            # use bisection to find the root
            newt_res = newton(bs_func, bs_vega_func, 0.3, True)
            if newt_res:
                k_ivs[(exp, K)] = newt_res

    return k_ivs
```

The below shows the results of using Newton's method to calculate implied volatility at the money, or as close to the money as possible. I am only using call options in Newtons.

In [13]:
```
%%time
iv_newton_atm, iv_newton_atm_Ks = calc_iv_newton_atm(data1); iv_newton_atm
```

```
CPU times: total: 0 ns
Wall time: 6.55 ms
```

Out[13]:
```
{('SPY', '2026-02-20'): 0.16067688597409663,
 ('SPY', '2026-03-20'): 0.1567329302525882,
 ('SPY', '2026-04-17'): 0.15099184107925245,
 ('TSLA', '2026-02-20'): 0.46049816898540047,
 ('TSLA', '2026-03-20'): 0.45203736418983526,
 ('TSLA', '2026-04-17'): 0.4552768832260244}
```

The below shows the results of using Newton's method to calculate implied volatility in the moneyness range of 0.9 to 1.1, or as close to the money as possible. I am only using call options in Newtons.

In [14]:
```
%%time
iv_newton_range, iv_newton_range_Ks, iv_newton_range_ivs = calc_iv_newton_range(data1, 0.9, 1.1); iv_newton_
```

```
CPU times: total: 172 ms
Wall time: 186 ms
```

Out[14]:
```
{('SPY', '2026-02-20'): 0.1584089883972207,
 ('SPY', '2026-03-20'): 0.15266280151826303,
 ('SPY', '2026-04-17'): 0.1401046167890857,
 ('TSLA', '2026-02-20'): 0.45883559115919653,
 ('TSLA', '2026-03-20'): 0.454712112058692,
 ('TSLA', '2026-04-17'): 0.4581763004483009}
```

In [15]:
```
diff = 0
for key in iv_newton_atm:
    diff += abs(iv_bisection_atm[key] - iv_newton_atm[key])
    diff += abs(iv_bisection_range[key] - iv_newton_range[key])

diff
```

Out[15]:
```
6.940791028864357e-07
```

Bisection vs. Newton

Comparing the CPU times of bisection vs Newton, there is a clear winner.

NOTE THESE TIMES ARE WHAT I GOT WHEN I RAN IT. RUNTIMES WILL DIFFER IF CELL IS RERUN.

With bisection, runtimes of:

- ATM: 20ms
- Range: 600ms

With Newton's, runtimes of:

- ATM: 5ms
- Range: 190ms

Newton's method was over 2x more efficient than bisection. Additionally, comparing the differences between the calculated implied volatilities, we see that there is only 2.14e-06 difference across all values combined. Barely any difference in the level of accuracy.

8

In [16]:
```
rows = []

for key in iv_newton_atm:
    ticker, expiry = key
    rows.append({
        'Ticker': ticker,
```

```
        'Expiry': expiry,
        'IV_ATM': iv_newton_atm[key],
        'IV_Avg': iv_newton_range.get(key, np.nan)
    })

iv_newton = pd.DataFrame(rows).sort_values(['Ticker', 'Expiry']); iv_newton
```

Out[16]:

	Ticker	Expiry	IV_ATM	IV_Avg
0	SPY	2026-02-20	0.160677	0.158409
1	SPY	2026-03-20	0.156733	0.152663
2	SPY	2026-04-17	0.150992	0.140105
3	TSLA	2026-02-20	0.460498	0.458836
4	TSLA	2026-03-20	0.452037	0.454712
5	TSLA	2026-04-17	0.455277	0.458176

In [17]:
```
vix = get_ohlcv(tickers=['^VIX'], start='2026-02-04', end='2026-02-05', interval='1m', save=False)
vix_vol = float(vix['Close'].iloc[-1]) / 100.0
print(f'^VIX Closing on {vix.index[0].date().isoformat()}: {vix_vol}')
```

```
^VIX Closing on 2026-02-04: 0.19290000915527344
```

TSLA vs SPY

The implied volatilities for TSLA are much higher than those for SPY. This is probably because SPY is an ETF that tracks the S&P 500, so it is very diverse. Currently, SPY has holdings in NVIDIA, Apple, Microsoft, Amazon, and Alphabet to name a few. Thus, the implied volatility for SPY is a culmination of all their holdings. Additionally, TSLA has higher implied volatilities recently because their most recent earnings report was on Jan 28.

Comparison with ^VIX

SPY's implied volatility is most similar to ^VIX's. This makes sense because VIX tracks the expected volatility using index options, while the implied volatility for SPY was calculated (in our case) using BS formulas and options.

Maturity Effect

As maturity increased, implied volatiility for SPY tended to decrease while TSLA tended to flatten. For SPY, it makes sense that the implied volatility (IV) decreases because of the market's stylized fact of mean reversion. I assume that TSLA's IV hovers around the same 45% because it's rise was very sudden and investors are still speculative of it.

ATM vs OTM & ITM

The ATM IV's for the most part, are lower than the ITM/OTM averages. With TSLA, a volatility smile can be observed with the local minimum being 0.452, however with SPY, the volatility is purely decreasing. The OTM/ITM pairs may be higher than expected because I averaged over the moneyness range of (0.9, 1.1).

9

In [18]:
```
def calc_put_parity(C, S0, K, r, T):
    return C - S0 + K * np.exp(-r * T)

def calc_call_parity(P, S0, K, r, T):
    return P + S0 - K * np.exp(-r * T)

contracts = {}
lo = 0.90
hi = 1.10
```

```
calls = data1[data1['type'] == 'call']
for _, call in calls.iterrows():
    ticker = call['ticker']
    S0 = float(call['S0'])
    exp = call['expiry']
    K = float(call['strike'])
    r = FED_FUNDS_RATE

    moneyness = S0 / K
    if not (lo <= moneyness <= hi):
        continue

    vol = iv_newton[(iv_newton['Expiry'] == exp) & (iv_newton['Ticker'] == ticker)]['IV_Avg'].iloc[0]

    now = call['S0_time']
    T = get_maturity(exp, now)

    market_call = 0.5 * (call['bid'] + call['ask'])
    bs_call = bs_formula('C', S0, vol, T, K, r)
    par_put = calc_put_parity(bs_call, S0, K, r, T)

    key = f'{ticker}_{exp}_{K:.2f}'

    contracts[key] = {
        'ticker': ticker,
        'expiry': exp,
        'strike': K,
        'S0': S0,
        'T': T,
        'vol': vol,
        'r': r,

        'moneyness': moneyness,

        'market_call': market_call,
        'market_put': np.nan,

        'bs_call': bs_call,
        'bs_put': np.nan,

        'par_call': np.nan,
        'par_put': par_put
    }


puts = data1[data1['type'] == 'put']
for _, put in puts.iterrows():
    ticker = put['ticker']
    S0 = float(put['S0'])
    exp = put['expiry']
    K = float(put['strike'])
    r = FED_FUNDS_RATE

    moneyness = S0 / K
    if not (lo <= moneyness <= hi):
        continue

    vol = iv_newton[(iv_newton['Expiry'] == exp) & (iv_newton['Ticker'] == ticker)]['IV_Avg'].iloc[0]

    now = put['S0_time']
    T = get_maturity(exp, now)

    market_put = 0.5 * (put['bid'] + put['ask'])
    bs_put = bs_formula('P', S0, vol, T, K, r)
    par_call = calc_call_parity(bs_put, S0, K, r, T)

    key = f'{ticker}_{exp}_{K:.2f}'

    if key not in contracts:
        contracts[key] = {
```

```
            'ticker': ticker,
            'expiry': exp,
            'strike': K,
            'S0': S0,
            'T': T,
            'vol': vol,
            'r': r,

            'market_call': np.nan,
            'market_put': market_put,

            'bs_call': np.nan,
            'bs_put': bs_put,

            'par_call': par_call,
            'par_put': np.nan
        }
    else:
        contracts[key].update({
            'market_put': market_put,
            'bs_put': bs_put,
            'par_call': par_call
        })


tsla_c = {}
spy_c = {}
for c in contracts:
    if contracts[c]['ticker'] == 'TSLA':
        tsla_c[c] = contracts[c]
    else:
        spy_c[c] = contracts[c]

tsla_df = pd.DataFrame.from_dict(tsla_c, orient='index').round(4)
spy_df = pd.DataFrame.from_dict(spy_c, orient='index').round(4)

def remake_df_trimmed(in_tsla=tsla_df, in_spy=spy_df):
    out_tsla = in_tsla[(0.95 <= tsla_df['moneyness']) & (tsla_df['moneyness'] <= 1.05)].copy()
    out_spy = in_spy[(0.99 <= spy_df['moneyness']) & (spy_df['moneyness'] <= 1.01)].copy()

    return out_tsla, out_spy

tsla_dft, spy_dft = remake_df_trimmed()
# tsla_dft = tsla_df[(0.95 <= tsla_df['moneyness']) & (tsla_df['moneyness'] <= 1.05)]
# spy_dft = spy_df[(0.99 <= spy_df['moneyness']) & (spy_df['moneyness'] <= 1.01)]
```

In [19]:
```
tsla_dft
```

Out[19]:

	ticker	expiry	strike	S0	T	vol	r	moneyness	market_call	market_put
TSLA_2026-02-20_385.00	TSLA	2026-02-20	385.0	404.2	0.0476	0.4588	0.0364	1.0499	28.375	8.450
TSLA_2026-02-20_390.00	TSLA	2026-02-20	390.0	404.2	0.0476	0.4588	0.0364	1.0364	24.675	10.025
TSLA_2026-02-20_395.00	TSLA	2026-02-20	395.0	404.2	0.0476	0.4588	0.0364	1.0233	21.525	11.850
TSLA_2026-02-20_400.00	TSLA	2026-02-20	400.0	404.2	0.0476	0.4588	0.0364	1.0105	18.850	13.850
TSLA_2026-02-20_405.00	TSLA	2026-02-20	405.0	404.2	0.0476	0.4588	0.0364	0.9980	16.150	16.250
TSLA_2026-02-20_410.00	TSLA	2026-02-20	410.0	404.2	0.0476	0.4588	0.0364	0.9859	13.650	19.025
TSLA_2026-02-20_412.50	TSLA	2026-02-20	412.5	404.2	0.0476	0.4588	0.0364	0.9799	12.450	20.475
TSLA_2026-02-20_415.00	TSLA	2026-02-20	415.0	404.2	0.0476	0.4588	0.0364	0.9740	11.600	21.725
TSLA_2026-02-20_417.50	TSLA	2026-02-20	417.5	404.2	0.0476	0.4588	0.0364	0.9681	10.500	23.425
TSLA_2026-02-20_420.00	TSLA	2026-02-20	420.0	404.2	0.0476	0.4588	0.0364	0.9624	9.650	25.100
TSLA_2026-02-20_422.50	TSLA	2026-02-20	422.5	404.2	0.0476	0.4588	0.0364	0.9567	8.900	26.750
TSLA_2026-02-20_425.00	TSLA	2026-02-20	425.0	404.2	0.0476	0.4588	0.0364	0.9511	8.125	28.150
TSLA_2026-03-20_385.00	TSLA	2026-03-20	385.0	404.2	0.1270	0.4547	0.0364	1.0499	38.025	16.700
TSLA_2026-03-20_390.00	TSLA	2026-03-20	390.0	404.2	0.1270	0.4547	0.0364	1.0364	34.800	18.675
TSLA_2026-03-20_395.00	TSLA	2026-03-20	395.0	404.2	0.1270	0.4547	0.0364	1.0233	31.575	20.775
TSLA_2026-03-20_400.00	TSLA	2026-03-20	400.0	404.2	0.1270	0.4547	0.0364	1.0105	29.100	23.175
TSLA_2026-03-20_405.00	TSLA	2026-03-20	405.0	404.2	0.1270	0.4547	0.0364	0.9980	26.450	25.575
TSLA_2026-03-20_410.00	TSLA	2026-03-20	410.0	404.2	0.1270	0.4547	0.0364	0.9859	24.025	27.950
TSLA_2026-03-20_415.00	TSLA	2026-03-20	415.0	404.2	0.1270	0.4547	0.0364	0.9740	21.875	31.000
TSLA_2026-03-20_420.00	TSLA	2026-03-20	420.0	404.2	0.1270	0.4547	0.0364	0.9624	19.825	33.950
TSLA_2026-03-20_425.00	TSLA	2026-03-20	425.0	404.2	0.1270	0.4547	0.0364	0.9511	17.725	36.800
TSLA_2026-04-17_385.00	TSLA	2026-04-17	385.0	404.2	0.2063	0.4582	0.0364	1.0499	45.075	23.075
TSLA_2026-04-17_390.00	TSLA	2026-04-17	390.0	404.2	0.2063	0.4582	0.0364	1.0364	42.350	25.300
TSLA_2026-04-17_395.00	TSLA	2026-04-17	395.0	404.2	0.2063	0.4582	0.0364	1.0233	39.725	27.400
TSLA_2026-04-17_400.00	TSLA	2026-04-17	400.0	404.2	0.2063	0.4582	0.0364	1.0105	36.900	29.975
TSLA_2026-04-17_405.00	TSLA	2026-04-17	405.0	404.2	0.2063	0.4582	0.0364	0.9980	34.325	32.200
TSLA_2026-04-17_410.00	TSLA	2026-04-17	410.0	404.2	0.2063	0.4582	0.0364	0.9859	32.275	34.800
TSLA_2026-04-17_415.00	TSLA	2026-04-17	415.0	404.2	0.2063	0.4582	0.0364	0.9740	29.725	37.625
TSLA_2026-04-17_420.00	TSLA	2026-04-17	420.0	404.2	0.2063	0.4582	0.0364	0.9624	27.625	40.400
TSLA_2026-04-17_425.00	TSLA	2026-04-17	425.0	404.2	0.2063	0.4582	0.0364	0.9511	25.875	43.475

In [20]:
```
spy_dft
```

Out[20]:

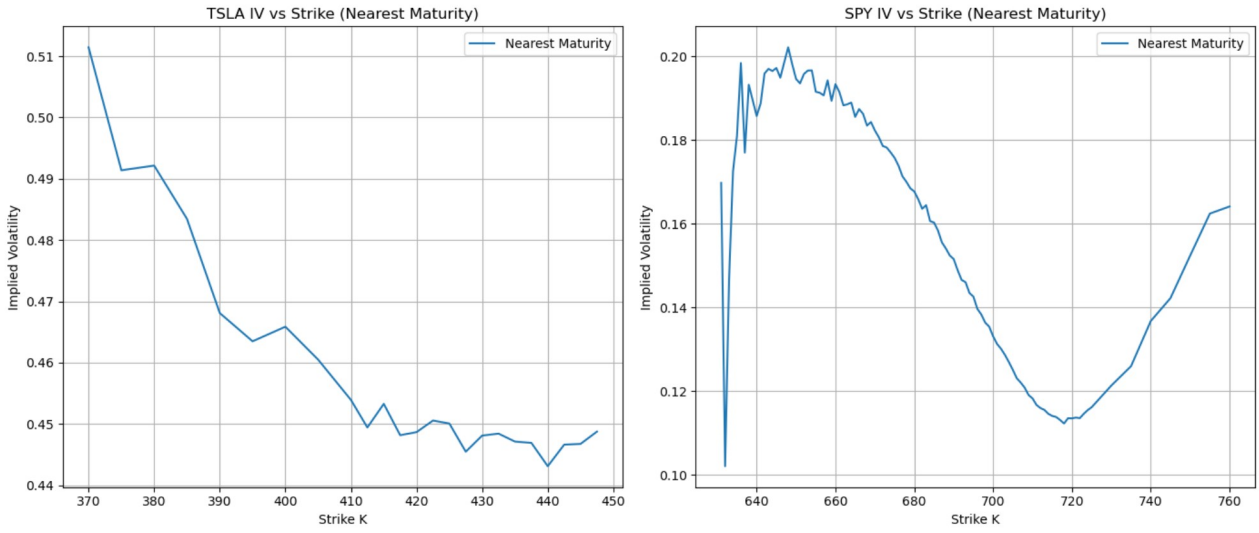
	ticker	expiry	strike	S0	T	vol	r	moneyness	market_call	market_put
SPY_2026-02-20_678.00	SPY	2026-02-20	678.0	684.325	0.0476	0.1584	0.0364	1.0093	14.265	8.390
SPY_2026-02-20_679.00	SPY	2026-02-20	679.0	684.325	0.0476	0.1584	0.0364	1.0078	13.575	8.665
SPY_2026-02-20_680.00	SPY	2026-02-20	680.0	684.325	0.0476	0.1584	0.0364	1.0064	12.945	8.995
SPY_2026-02-20_681.00	SPY	2026-02-20	681.0	684.325	0.0476	0.1584	0.0364	1.0049	12.265	9.360
SPY_2026-02-20_682.00	SPY	2026-02-20	682.0	684.325	0.0476	0.1584	0.0364	1.0034	11.575	9.725
SPY_2026-02-20_683.00	SPY	2026-02-20	683.0	684.325	0.0476	0.1584	0.0364	1.0019	11.085	10.120
SPY_2026-02-20_684.00	SPY	2026-02-20	684.0	684.325	0.0476	0.1584	0.0364	1.0005	10.335	10.465
SPY_2026-02-20_685.00	SPY	2026-02-20	685.0	684.325	0.0476	0.1584	0.0364	0.9990	9.805	10.885
SPY_2026-02-20_686.00	SPY	2026-02-20	686.0	684.325	0.0476	0.1584	0.0364	0.9976	9.200	11.215
SPY_2026-02-20_687.00	SPY	2026-02-20	687.0	684.325	0.0476	0.1584	0.0364	0.9961	8.555	11.645
SPY_2026-02-20_688.00	SPY	2026-02-20	688.0	684.325	0.0476	0.1584	0.0364	0.9947	8.010	12.130
SPY_2026-02-20_689.00	SPY	2026-02-20	689.0	684.325	0.0476	0.1584	0.0364	0.9932	7.470	12.560
SPY_2026-02-20_690.00	SPY	2026-02-20	690.0	684.325	0.0476	0.1584	0.0364	0.9918	6.995	13.190
SPY_2026-02-20_691.00	SPY	2026-02-20	691.0	684.325	0.0476	0.1584	0.0364	0.9903	6.435	13.510
SPY_2026-03-20_678.00	SPY	2026-03-20	678.0	684.325	0.1270	0.1527	0.0364	1.0093	20.975	13.815
SPY_2026-03-20_679.00	SPY	2026-03-20	679.0	684.325	0.1270	0.1527	0.0364	1.0078	20.275	14.145
SPY_2026-03-20_680.00	SPY	2026-03-20	680.0	684.325	0.1270	0.1527	0.0364	1.0064	19.670	14.475
SPY_2026-03-20_681.00	SPY	2026-03-20	681.0	684.325	0.1270	0.1527	0.0364	1.0049	18.980	14.875
SPY_2026-03-20_682.00	SPY	2026-03-20	682.0	684.325	0.1270	0.1527	0.0364	1.0034	18.410	15.190
SPY_2026-03-20_683.00	SPY	2026-03-20	683.0	684.325	0.1270	0.1527	0.0364	1.0019	17.660	15.560
SPY_2026-03-20_684.00	SPY	2026-03-20	684.0	684.325	0.1270	0.1527	0.0364	1.0005	17.010	15.915
SPY_2026-03-20_685.00	SPY	2026-03-20	685.0	684.325	0.1270	0.1527	0.0364	0.9990	16.380	16.305
SPY_2026-03-20_686.00	SPY	2026-03-20	686.0	684.325	0.1270	0.1527	0.0364	0.9976	15.740	16.660
SPY_2026-03-20_687.00	SPY	2026-03-20	687.0	684.325	0.1270	0.1527	0.0364	0.9961	15.130	17.045
SPY_2026-03-20_688.00	SPY	2026-03-20	688.0	684.325	0.1270	0.1527	0.0364	0.9947	14.525	17.480
SPY_2026-03-20_689.00	SPY	2026-03-20	689.0	684.325	0.1270	0.1527	0.0364	0.9932	13.940	17.890
SPY_2026-03-20_690.00	SPY	2026-03-20	690.0	684.325	0.1270	0.1527	0.0364	0.9918	13.335	18.355
SPY_2026-03-20_691.00	SPY	2026-03-20	691.0	684.325	0.1270	0.1527	0.0364	0.9903	12.770	18.650
SPY_2026-04-17_678.00	SPY	2026-04-17	678.0	684.325	0.2063	0.1401	0.0364	1.0093	25.530	17.415
SPY_2026-04-17_679.00	SPY	2026-04-17	679.0	684.325	0.2063	0.1401	0.0364	1.0078	24.790	17.775
SPY_2026-04-17_680.00	SPY	2026-04-17	680.0	684.325	0.2063	0.1401	0.0364	1.0064	24.215	18.095
SPY_2026-04-17_681.00	SPY	2026-04-17	681.0	684.325	0.2063	0.1401	0.0364	1.0049	23.425	18.365
SPY_2026-04-17_682.00	SPY	2026-04-17	682.0	684.325	0.2063	0.1401	0.0364	1.0034	22.890	18.780
SPY_2026-04-17_683.00	SPY	2026-04-17	683.0	684.325	0.2063	0.1401	0.0364	1.0019	22.135	19.135
SPY_2026-04-17_684.00	SPY	2026-04-17	684.0	684.325	0.2063	0.1401	0.0364	1.0005	21.495	19.495
SPY_2026-04-17_685.00	SPY	2026-04-17	685.0	684.325	0.2063	0.1401	0.0364	0.9990	20.845	19.870
SPY_2026-04-17_686.00	SPY	2026-04-17	686.0	684.325	0.2063	0.1401	0.0364	0.9976	20.225	20.255
SPY_2026-04-17_687.00	SPY	2026-04-17	687.0	684.325	0.2063	0.1401	0.0364	0.9961	19.660	20.490

	ticker	expiry	strike	S0	T	vol	r	moneyness	market_call	market_put
SPY_2026-04-17_688.00	SPY	2026-04-17	688.0	684.325	0.2063	0.1401	0.0364	0.9947	19.000	21.060
SPY_2026-04-17_689.00	SPY	2026-04-17	689.0	684.325	0.2063	0.1401	0.0364	0.9932	18.395	21.435
SPY_2026-04-17_690.00	SPY	2026-04-17	690.0	684.325	0.2063	0.1401	0.0364	0.9918	17.795	21.860
SPY_2026-04-17_691.00	SPY	2026-04-17	691.0	684.325	0.2063	0.1401	0.0364	0.9903	17.220	22.285

The above tables shows the options that are in 0.95-1.05 moneyness for TSLA and 0.99-1.01 moneyness for SPY. Market call and put refer to the mid point from actual option data. BS call and put refer to the prices calculated from BS formulas. Par call and put refer to the values calculated from the CP parity.

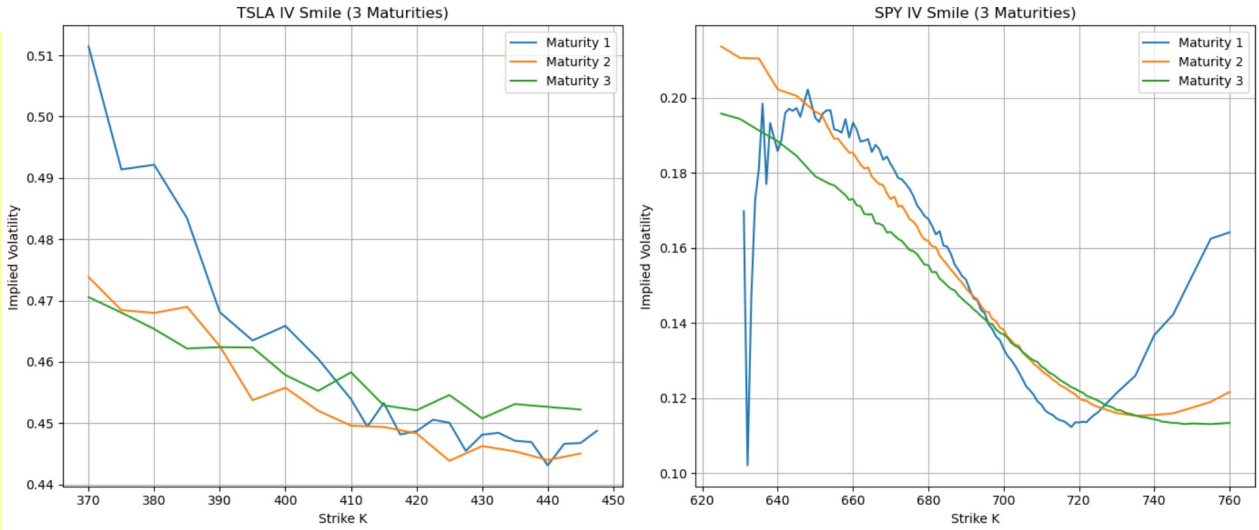
10

In [21]:
```
spy_ivs = iv_newton_range_ivs[0:3]
spy_Ks = iv_newton_range_Ks[0:3]
tsla_ivs = iv_newton_range_ivs[3:]
tsla_Ks = iv_newton_range_Ks[3:]

fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# tsla plot
ax[0].plot(tsla_Ks[0], tsla_ivs[0], label='Nearest Maturity')
ax[0].set_xlabel('Strike K')
ax[0].set_ylabel('Implied Volatility')
ax[0].set_title('TSLA IV vs Strike (Nearest Maturity)')
ax[0].legend()
ax[0].grid(True)

# spy plot
ax[1].plot(spy_Ks[0], spy_ivs[0], label='Nearest Maturity')
ax[1].set_xlabel('Strike K')
ax[1].set_ylabel('Implied Volatility')
ax[1].set_title('SPY IV vs Strike (Nearest Maturity)')
ax[1].legend()
ax[1].grid(True)

plt.tight_layout()
plt.show()
```



The above shows the plots of TSLA and SPY's implied volatilities vs. strike K for the closest to maturity options, which in this case are 2026-02-20 from spot date of 2026-02-04. In both cases we see a volatility smile. SPY's is more prominent where we can also see the local min. There also seems to be some bad data or weird spike from strikes 620-650.

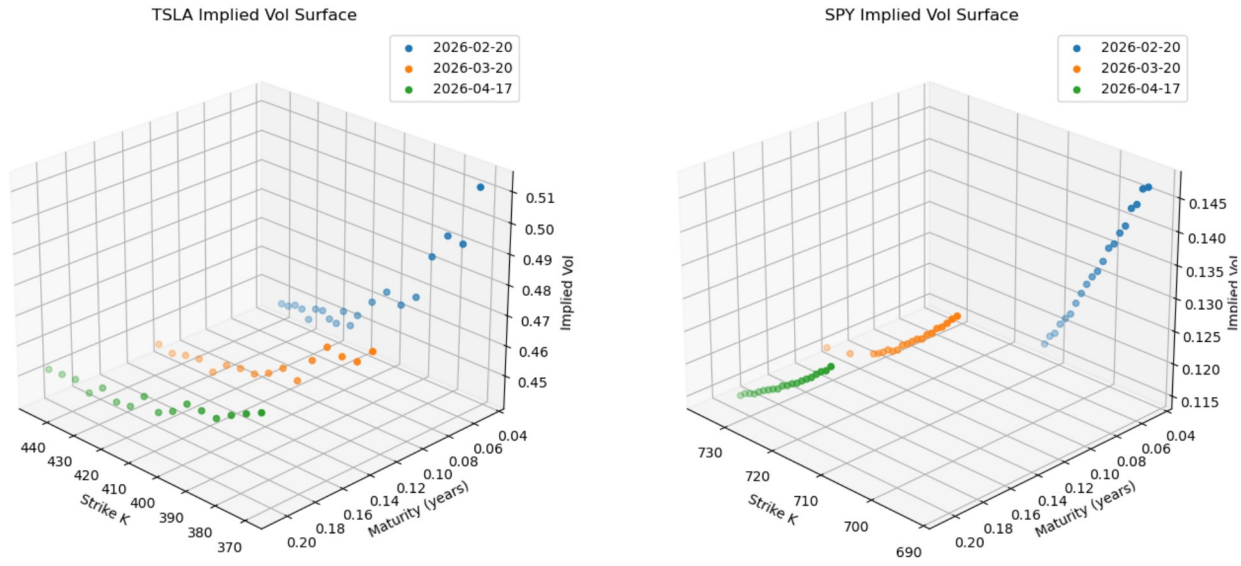
In [22]:
```
fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# tsla plot
for i in range(3):
    ax[0].plot(tsla_Ks[i], tsla_ivs[i], label=f'Maturity {i+1}')

ax[0].set_xlabel('Strike K')
ax[0].set_ylabel('Implied Volatility')
ax[0].set_title('TSLA IV Smile (3 Maturities)')
ax[0].legend()
ax[0].grid(True)

# spy plot
for i in range(3):
    ax[1].plot(spy_Ks[i], spy_ivs[i], label=f'Maturity {i+1}')

ax[1].set_xlabel('Strike K')
ax[1].set_ylabel('Implied Volatility')
ax[1].set_title('SPY IV Smile (3 Maturities)')
ax[1].legend()
ax[1].grid(True)

plt.tight_layout()
plt.show()
```



The above shows the plots of TSLA and SPY's implied volatilities vs. strike K for all three maturities. Now that weird spike for SPY is a little more concerning. Generally, the blue line should be slightly higher than orange (maturity 2), but for SPY, it is very clearly opposite. The TSLA IV's look good and as expected.

In [23]:
```
fig = plt.figure(figsize=(14,6))

# tsla plot
ticker = 'TSLA'
exps = sorted(data1[data1['ticker']==ticker]['expiry'].unique())[:3]
taus = [get_maturity(exp, data1['S0_time'].iloc[0]) for exp in exps]

ax1 = fig.add_subplot(121, projection='3d')

for i in range(3):
    Ks  = np.array(tsla_Ks[i])[:20]
    ivs = np.array(tsla_ivs[i])[:20]
    T   = np.full_like(Ks, fill_value=taus[i], dtype=float)

    ax1.scatter(Ks, T, ivs, label=f'{exps[i]}')

ax1.set_xlabel('Strike K')
ax1.set_ylabel('Maturity (years)')
ax1.set_zlabel('Implied Vol')
```

```
ax1.set_title('TSLA Implied Vol Surface')
ax1.view_init(elev=25, azim=135)
ax1.legend()


# spy plot
ticker = 'SPY'
exps = sorted(data1[data1['ticker']==ticker]['expiry'].unique())[:3]
taus = [get_maturity(exp, data1['S0_time'].iloc[0]) for exp in exps]

ax2 = fig.add_subplot(122, projection='3d')

for i in range(3):
    Ks  = np.array(spy_Ks[i])[60:80]
    ivs = np.array(spy_ivs[i])[60:80]
    T   = np.full_like(Ks, fill_value=taus[i], dtype=float)

    ax2.scatter(Ks, T, ivs, label=f'{exps[i]}')

ax2.set_xlabel('Strike K')
ax2.set_ylabel('Maturity (years)')
ax2.set_zlabel('Implied Vol')
ax2.set_title('SPY Implied Vol Surface')
ax2.view_init(elev=25, azim=135)
ax2.legend()

plt.tight_layout()
plt.show()
```



The above shows the plots of TSLA and SPY's implied volatilities vs. strike K for the all maturities on a 3D plot. I chose to show the local minima portions for SPY because the first 20 values were a little wonky for the first maturity.

11

In [24]:
```
# bs formulas for greeks
def bs_delta(S0, vol, T, K, r):
    d1 = (np.log(S0 / K) + (r + vol**2 / 2) * T) / (vol * T**0.5)

    return norm.cdf(d1)

def bs_gamma(S0, vol, T, K, r):
    d1 = (np.log(S0 / K) + (r + vol**2 / 2) * T) / (vol * T**0.5)

    return norm.pdf(d1) / (S0 * vol * T**0.5)

def bs_greeks(S0, vol, T, K, r):
```

```
    delta = bs_delta(S0, vol, T, K, r)
    gamma = bs_gamma(S0, vol, T, K, r)
    vega = bs_vega(S0, vol, T, K, r)

    return delta, gamma, vega

# finite difference for greeks
def fd_greeks(S0, vol, T, K, r, cp='C'):
    percent = 1e-3
    step = percent * S0

    P_0 = bs_formula(cp, S0, vol, T, K, r)
    # price increase by step
    P_u = bs_formula(cp, S0 + step, vol, T, K, r)
    # price decrease by step
    P_d = bs_formula(cp, S0 - step, vol, T, K, r)
    # vol increase by step percent
    P_uv = bs_formula(cp, S0, vol + percent, T, K, r)
    # vol decrease by step percent
    P_dv = bs_formula(cp, S0, vol - percent, T, K, r)

    delta = (P_u - P_d) / (2.0 * step)
    gamma = (P_u - 2 * P_0 + P_d) / (step**2)
    vega = (P_uv - P_dv) / (2.0 * percent)

    return delta, gamma, vega

# make sure we have correct trimmed dfs range(0.9, 1.1) and (0.95, 1.05)
tsla_dft, spy_dft = remake_df_trimmed()

for i, row in tsla_dft.iterrows():
    S0 = row['S0']
    vol = row['vol']
    T = row['T']
    K = row['strike']
    r = row['r']

    # calc greeks
    bs_d, bs_g, bs_v = bs_greeks(S0, vol, T, K, r)
    fd_d, fd_g, fd_v = fd_greeks(S0, vol, T, K, r)

    # add to dft
    tsla_dft.loc[i, 'bs_delta'] = bs_d
    tsla_dft.loc[i, 'bs_gamma'] = bs_g
    tsla_dft.loc[i, 'bs_vega'] = bs_v
    tsla_dft.loc[i, 'fd_delta'] = fd_d
    tsla_dft.loc[i, 'fd_gamma'] = fd_g
    tsla_dft.loc[i, 'fd_vega'] = fd_v

for i, row in spy_dft.iterrows():
    S0 = row['S0']
    vol = row['vol']
    T = row['T']
    K = row['strike']
    r = row['r']

    # calc greeks
    bs_d, bs_g, bs_v = bs_greeks(S0, vol, T, K, r)
    fd_d, fd_g, fd_v = fd_greeks(S0, vol, T, K, r)

    # add to dft
    spy_dft.loc[i, 'bs_delta'] = bs_d
    spy_dft.loc[i, 'bs_gamma'] = bs_g
    spy_dft.loc[i, 'bs_vega'] = bs_v
    spy_dft.loc[i, 'fd_delta'] = fd_d
    spy_dft.loc[i, 'fd_gamma'] = fd_g
    spy_dft.loc[i, 'fd_vega'] = fd_v

def remake_greek_dft(in_tsla=tsla_dft, in_spy=spy_dft):
    out_tsla = in_tsla[['bs_delta', 'bs_gamma', 'bs_vega', 'fd_delta', 'fd_gamma', 'fd_vega']].copy()
```

```
    out_spy = in_spy[['bs_delta', 'bs_gamma', 'bs_vega', 'fd_delta', 'fd_gamma', 'fd_vega']].copy()

    return out_tsla, out_spy

tsla_g_dft, spy_g_dft = remake_greek_dft()
```

In [25]:
```
tsla_g_dft
```

Out[25]:

	bs_delta	bs_gamma	bs_vega	fd_delta	fd_gamma	fd_vega
TSLA_2026-02-20_385.00	0.710055	0.008460	30.183756	0.710051	0.008460	30.183740
TSLA_2026-02-20_390.00	0.664450	0.009010	32.148036	0.664446	0.009010	32.148026
TSLA_2026-02-20_395.00	0.616909	0.009434	33.659480	0.616907	0.009434	33.659475
TSLA_2026-02-20_400.00	0.568167	0.009716	34.666303	0.568165	0.009716	34.666301
TSLA_2026-02-20_405.00	0.518985	0.009849	35.141293	0.518984	0.009849	35.141293
TSLA_2026-02-20_410.00	0.470117	0.009833	35.082393	0.470117	0.009832	35.082392
TSLA_2026-02-20_412.50	0.446027	0.009770	34.858673	0.446027	0.009770	34.858671
TSLA_2026-02-20_415.00	0.422278	0.009673	34.511354	0.422278	0.009672	34.511349
TSLA_2026-02-20_417.50	0.398948	0.009542	34.046386	0.398949	0.009542	34.046379
TSLA_2026-02-20_420.00	0.376110	0.009381	33.470766	0.376111	0.009381	33.470756
TSLA_2026-02-20_422.50	0.353829	0.009191	32.792389	0.353831	0.009191	32.792376
TSLA_2026-02-20_425.00	0.332165	0.008974	32.019886	0.332167	0.008974	32.019869
TSLA_2026-03-20_385.00	0.659054	0.005600	52.835612	0.659053	0.005600	52.835598
TSLA_2026-03-20_390.00	0.629395	0.005768	54.415766	0.629394	0.005768	54.415758
TSLA_2026-03-20_395.00	0.599339	0.005901	55.674750	0.599338	0.005901	55.674746
TSLA_2026-03-20_400.00	0.569071	0.005999	56.602180	0.569070	0.005999	56.602179
TSLA_2026-03-20_405.00	0.538775	0.006062	57.193997	0.538774	0.006062	57.193997
TSLA_2026-03-20_410.00	0.508627	0.006090	57.452203	0.508626	0.006090	57.452202
TSLA_2026-03-20_415.00	0.478795	0.006082	57.384440	0.478795	0.006082	57.384438
TSLA_2026-03-20_420.00	0.449436	0.006042	57.003442	0.449436	0.006042	57.003436
TSLA_2026-03-20_425.00	0.420693	0.005970	56.326389	0.420693	0.005970	56.326378
TSLA_2026-04-17_385.00	0.645792	0.004422	68.294378	0.645791	0.004422	68.294366
TSLA_2026-04-17_390.00	0.622473	0.004517	69.762233	0.622472	0.004517	69.762225
TSLA_2026-04-17_395.00	0.599005	0.004596	70.974186	0.599004	0.004596	70.974182
TSLA_2026-04-17_400.00	0.575475	0.004657	71.926632	0.575474	0.004657	71.926630
TSLA_2026-04-17_405.00	0.551970	0.004702	72.618957	0.551969	0.004702	72.618956
TSLA_2026-04-17_410.00	0.528572	0.004730	73.053367	0.528572	0.004730	73.053366
TSLA_2026-04-17_415.00	0.505361	0.004742	73.234678	0.505360	0.004742	73.234676
TSLA_2026-04-17_420.00	0.482409	0.004738	73.170080	0.482409	0.004738	73.170075
TSLA_2026-04-17_425.00	0.459786	0.004718	72.868876	0.459786	0.004718	72.868868

In [26]:
```
spy_g_dft
```

Out[26]:

	bs_delta	bs_gamma	bs_vega	fd_delta	fd_gamma	fd_vega
SPY_2026-02-20_678.00	0.631605	0.015943	56.291816	0.631585	0.015942	56.291706
SPY_2026-02-20_679.00	0.615415	0.016158	57.052610	0.615397	0.016157	57.052526
SPY_2026-02-20_680.00	0.599045	0.016346	57.717704	0.599029	0.016345	57.717643
SPY_2026-02-20_681.00	0.582523	0.016507	58.284000	0.582510	0.016506	58.283958
SPY_2026-02-20_682.00	0.565880	0.016638	58.748921	0.565869	0.016637	58.748896
SPY_2026-02-20_683.00	0.549143	0.016741	59.110435	0.549134	0.016740	59.110422
SPY_2026-02-20_684.00	0.532343	0.016814	59.367064	0.532337	0.016812	59.367059
SPY_2026-02-20_685.00	0.515510	0.016856	59.517896	0.515506	0.016855	59.517895
SPY_2026-02-20_686.00	0.498674	0.016869	59.562586	0.498672	0.016868	59.562586
SPY_2026-02-20_687.00	0.481865	0.016852	59.501361	0.481865	0.016850	59.501357
SPY_2026-02-20_688.00	0.465112	0.016804	59.335007	0.465115	0.016803	59.334994
SPY_2026-02-20_689.00	0.448445	0.016728	59.064862	0.448450	0.016727	59.064836
SPY_2026-02-20_690.00	0.431892	0.016623	58.692798	0.431900	0.016621	58.692756
SPY_2026-02-20_691.00	0.415482	0.016489	58.221203	0.415492	0.016488	58.221143
SPY_2026-03-20_678.00	0.611333	0.010293	93.477769	0.611326	0.010293	93.477641
SPY_2026-03-20_679.00	0.600913	0.010368	94.161939	0.600907	0.010368	94.161835
SPY_2026-03-20_680.00	0.590436	0.010436	94.780702	0.590430	0.010436	94.780620
SPY_2026-03-20_681.00	0.579910	0.010497	95.333016	0.579905	0.010497	95.332954
SPY_2026-03-20_682.00	0.569342	0.010551	95.817987	0.569337	0.010550	95.817942
SPY_2026-03-20_683.00	0.558739	0.010597	96.234867	0.558735	0.010596	96.234836
SPY_2026-03-20_684.00	0.548110	0.010635	96.583055	0.548106	0.010635	96.583037
SPY_2026-03-20_685.00	0.537462	0.010666	96.862105	0.537459	0.010665	96.862095
SPY_2026-03-20_686.00	0.526803	0.010689	97.071717	0.526800	0.010688	97.071714
SPY_2026-03-20_687.00	0.516139	0.010704	97.211746	0.516137	0.010704	97.211745
SPY_2026-03-20_688.00	0.505480	0.010712	97.282193	0.505479	0.010712	97.282192
SPY_2026-03-20_689.00	0.494833	0.010712	97.283212	0.494832	0.010712	97.283208
SPY_2026-03-20_690.00	0.484204	0.010704	97.215101	0.484204	0.010704	97.215091
SPY_2026-03-20_691.00	0.473602	0.010689	97.078305	0.473602	0.010689	97.078287
SPY_2026-04-17_678.00	0.616289	0.008769	118.694009	0.616283	0.008769	118.693802
SPY_2026-04-17_679.00	0.607415	0.008827	119.477793	0.607410	0.008827	119.477619
SPY_2026-04-17_680.00	0.598498	0.008881	120.201232	0.598493	0.008881	120.201088
SPY_2026-04-17_681.00	0.589542	0.008930	120.863458	0.589537	0.008929	120.863342
SPY_2026-04-17_682.00	0.580553	0.008974	121.463701	0.580548	0.008974	121.463610
SPY_2026-04-17_683.00	0.571534	0.009014	122.001293	0.571531	0.009014	122.001224
SPY_2026-04-17_684.00	0.562492	0.009049	122.475665	0.562488	0.009049	122.475615
SPY_2026-04-17_685.00	0.553430	0.009079	122.886348	0.553427	0.009079	122.886314
SPY_2026-04-17_686.00	0.544353	0.009105	123.232977	0.544350	0.009105	123.232956
SPY_2026-04-17_687.00	0.535267	0.009126	123.515289	0.535264	0.009125	123.515278

	bs_delta	bs_gamma	bs_vega	fd_delta	fd_gamma	fd_vega
SPY_2026-04-17_688.00	0.526175	0.009142	123.733122	0.526173	0.009141	123.733118
SPY_2026-04-17_689.00	0.517083	0.009153	123.886417	0.517081	0.009153	123.886415
SPY_2026-04-17_690.00	0.507995	0.009160	123.975214	0.507993	0.009159	123.975213
SPY_2026-04-17_691.00	0.498916	0.009161	123.999657	0.498915	0.009161	123.999652

The above tables show the calculated greeks. Columns that begin with 'bs_..' refer to the greeks calculated via Black Scholes Formulas. Columns that begin with 'fd..' refer to the greeks calculated via finite difference methods.

12

In [27]:
```
# calc ivs for data1
df1_ivs = calc_ivs_newton(data1)

for i, row in data2.iterrows():
    bid = row['bid']
    ask = row['ask']
    mid = (bid + ask) / 2.0

    K = row['strike']
    S0 = row['S0']
    exp = row['expiry']
    now = row['S0_time']
    T = get_maturity(exp, now)
    if (exp, K) not in df1_ivs: continue
    # get ivs from data1
    vol = df1_ivs[(exp, K)]
    r = FED_FUNDS_RATE

    # calc bs prices
    # PS, i know i am calculating bs prices twice, once for calls then again for puts.... im too lazy to opt
    bs_call = round(bs_formula('C', S0, vol, T, K, r), 4)
    bs_put = round(bs_formula('P', S0, vol, T, K, r), 4)

    data2.loc[i, 'mid'] = mid
    data2.loc[i, 'bs_call'] = bs_call
    data2.loc[i, 'bs_put'] = bs_put
# data2_res = data2[['contractSymbol', 'strike', 'type', 'mid', 'bs_call', 'bs_put']].copy()
```

```
C:\Users\Steven\AppData\Local\Temp\ipykernel_26468\2011442916.py:3: RuntimeWarning: overflow encountered in s
calar power
  d1 = (np.log(S0 / K) + (r + vol**2 / 2) * T) / (vol * T**0.5)
C:\Users\Steven\AppData\Local\Temp\ipykernel_26468\3361329082.py:28: RuntimeWarning: overflow encountered in
scalar power
  d1 = (np.log(S0 / K) + (r + vol**2 / 2) * T) / (vol * T**0.5)
```

In [28]:
```
data2
```

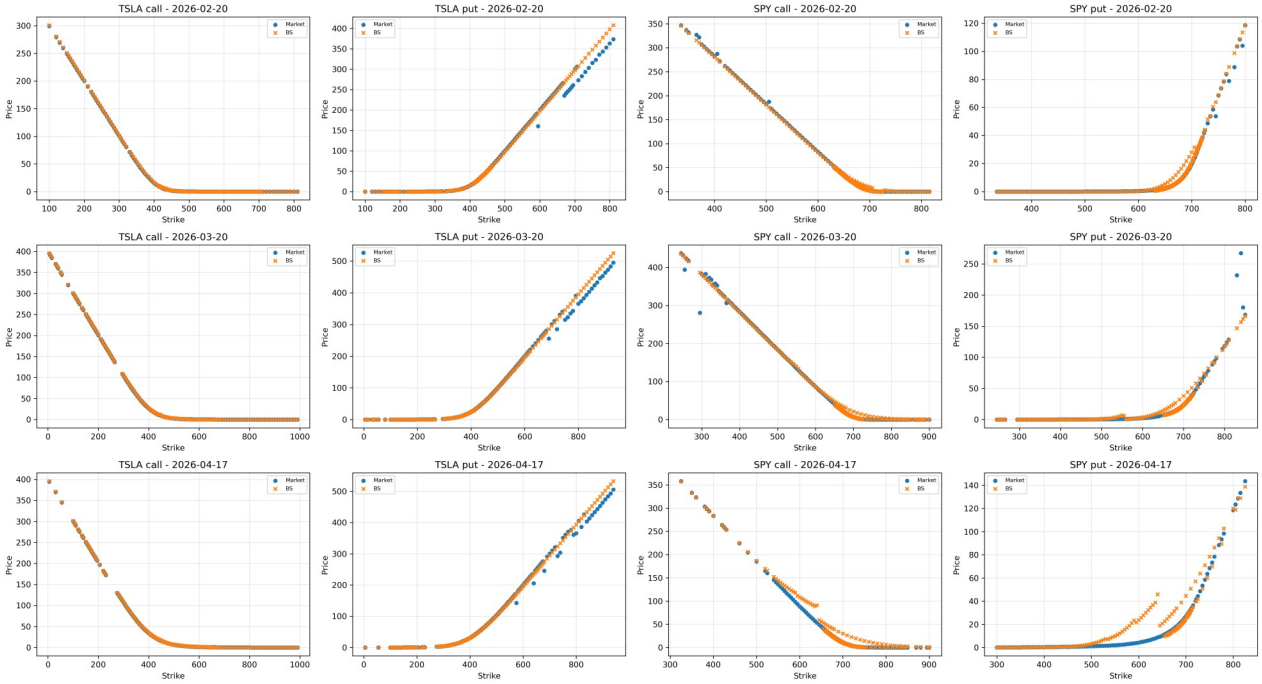
Out[28]:

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openI
0	TSLA260220C00100000	2026-02-05 17:26:26+00:00	100.0	300.30	298.95	300.00	-7.650024	-2.484177	77.0	
1	TSLA260220C00110000	2025-11-07 14:45:01+00:00	110.0	321.40	325.70	328.30	0.000000	0.000000	1.0	
2	TSLA260220C00120000	2026-01-16 17:08:41+00:00	120.0	319.63	278.50	280.55	0.000000	0.000000	90.0	
3	TSLA260220C00130000	2026-01-16 16:40:36+00:00	130.0	309.87	268.50	270.60	0.000000	0.000000	5.0	
4	TSLA260220C00140000	2026-01-16 20:53:02+00:00	140.0	299.78	258.50	260.60	0.000000	0.000000	6.0	
...	...	...	...	...	...	...	...	...	...	...
1949	SPY260417P00800000	2026-02-04 20:50:24+00:00	800.0	110.52	117.22	119.98	0.000000	0.000000	2.0	
1950	SPY260417P00805000	2026-02-04 20:50:24+00:00	805.0	115.54	122.12	124.93	0.000000	0.000000	2.0	
1951	SPY260417P00810000	2026-01-29 16:20:40+00:00	810.0	121.76	127.22	130.03	0.000000	0.000000	NaN	
1952	SPY260417P00815000	2026-01-29 16:20:16+00:00	815.0	127.17	132.00	135.17	0.000000	0.000000	NaN	
1953	SPY260417P00825000	2026-01-29 16:20:33+00:00	825.0	136.91	142.21	145.02	0.000000	0.000000	NaN	

1954 rows × 22 columns

In [29]:

```
df_plot = data2.copy()
df_plot['type'] = df_plot['type'].str.lower()
df_plot['expiry'] = pd.to_datetime(df_plot['expiry'])
expiries = [pd.Timestamp('2026-02-20'), pd.Timestamp('2026-03-20'), pd.Timestamp('2026-04-17')]

def plot_panel(ax, dft, ticker, opt_type, expiry):
    d = dft[(dft['ticker'] == ticker) & (dft['type'] == opt_type) & (dft['expiry'] == expiry)].copy()

    if opt_type == 'call':
        d = d.dropna(subset=['strike','mid','bs_call'])
        ax.scatter(d['strike'], d['mid'], s=18, label='Market')
        ax.scatter(d['strike'], d['bs_call'], s=18, marker='x', label='BS')
    else:
        d = d.dropna(subset=['strike','mid','bs_put'])
        ax.scatter(d['strike'], d['mid'], s=18, label='Market')
        ax.scatter(d['strike'], d['bs_put'], s=18, marker='x', label='BS')

    ax.set_title(f'{ticker} {opt_type} - {expiry.date()}')
    ax.set_xlabel('Strike')
    ax.set_ylabel('Price')
    ax.grid(True, alpha=0.3)
    ax.legend(fontsize=7)

fig, axes = plt.subplots(3, 4, figsize=(22, 12), dpi=300)
for i, exp in enumerate(expiries):
    plot_panel(axes[i,0], df_plot, 'TSLA', 'call', exp)
    plot_panel(axes[i,1], df_plot, 'TSLA', 'put',  exp)
    plot_panel(axes[i,2], df_plot, 'SPY',  'call', exp)
    plot_panel(axes[i,3], df_plot, 'SPY',  'put',  exp)

plt.tight_layout()
plt.show()
```



I plotted the results to better visualize how well I did. Each plot displays results for BS vs Market for the respective asset, option type, and expiry. The TSLA results look very good. For the most part, BS did a good job pricing the option. And as expected, BS is, on average the upper bound of the price. The SPY results look very weird. Firstly, SPY's implied volatilities for the closest expiry was very wonky and was very volatile itself. However, after seeing the comparisons, BS did a good job of pricing and makes me believe those wonky volatilites are real and not something wrong with my code. For the furthest expiry on SPY, the values look very weird. It's hard to believe the code is wrong here because it is the only weird one out of 6 total plots. I genuinely am not sure what went wrong here. The BS prices look jagged?

Part 3

a)

Answer given from my handwritten solution. Please scroll to the end of the jupyter notebook pdf.

b)

In [30]:
```python
def trapz_func(f, a, b, n):

    h = (b - a) / n
    res = 0.0

    for i in range(n):
        left = a + i * h
        right = a + (i + 1) * h

        res += 0.5 * (f(left) + f(right)) * h

    return res


# case 1 dx and dy from functions found in a
def case1_dxy(s, x, y, gamma):
    k = x * y

    dx = x - (k / ((1 - gamma) * s))**0.5
    dy = ((k * (1 - gamma) * s)**0.5 - y) / (1 - gamma)
```

```
    return dx, dy


# case 2 dx and dy from functions found in a
def case2_dxy(s, x, y, gamma):
    k = x * y

    dx = (((k * (1 - gamma)) / s)**0.5 - x) / (1 - gamma)
    dy = y - ((k * s) / (1 - gamma))**0.5

    return dx, dy


def R_func(s, x, y, gamma):
    P = y / x

    case1 = P / (1 - gamma)
    case2 = P * (1 - gamma)

    if s > case1:
        _, dy = case1_dxy(s, x, y, gamma)

        return gamma * dy

    elif s < case2:
        dx, _ = case2_dxy(s, x, y, gamma)

        return gamma * dx * s

    else:
        return 0.0

# f_S(t+1)(s) lognormal density function
def log_density_func(s, vol, dt):
    if s <= 0.0: return 0.0

    mu = -0.5 * vol**2 * dt
    var = vol**2 * dt
    res = (1.0 / (s * (2.0 * np.pi * var)**0.5)) * np.exp(-(np.log(s) - mu)**2 / (2.0 * var))

    return res

def expected_revenue(x, y, gamma, vol, dt, a=0.1, b=2.0, n=10000):
    def integrand(s):
        return R_func(s, x, y, gamma) * log_density_func(s, vol, dt)

    return trapz_func(integrand, a, b, n)


x_t, y_t = 1000.0, 1000.0
dt = 1.0 / 365.0
vol = 0.2
gamma = 0.003 # 30 bp

exp_rev = expected_revenue(x_t, y_t, gamma, vol, dt)
print(f'E[R(S)]: {exp_rev}')
```

```
E[R(S)]: 0.008521970586500997
```

c)

In [31]:

```
vols = [0.2, 0.6, 1.0]
gammas = [0.001, 0.003, 0.01]
x_t, y_t = 1000.0, 1000.0
dt = 1.0 / 365.0
vol = 0.2
gamma = 0.003 # 30 bp
```

```
er_tbl = []
for v in vols:
    for g in gammas:
        er = expected_revenue(x_t, y_t, g, v, dt)
        er_tbl.append({'vol': v, 'gamma': g, 'E[R]': er})

er_tbl = pd.DataFrame(er_tbl)
er_tbl = er_tbl.pivot(index='vol', columns='gamma', values='E[R]').sort_index()

gamma_star = er_tbl.idxmax(axis=1)
er_max = er_tbl.max(axis=1)
bests = pd.DataFrame({
    'gamma_star': gamma_star,
    'E[R]_max': er_max
})
```

In [32]:
```
er_tbl
```

Out[32]:

gamma	0.001	0.003	0.010
vol			
0.2	0.003685	0.008522	0.009430
0.6	0.011923	0.032983	0.081082
1.0	0.020061	0.057384	0.160690

The above table shows the E[R] values for each vol, gamma input.

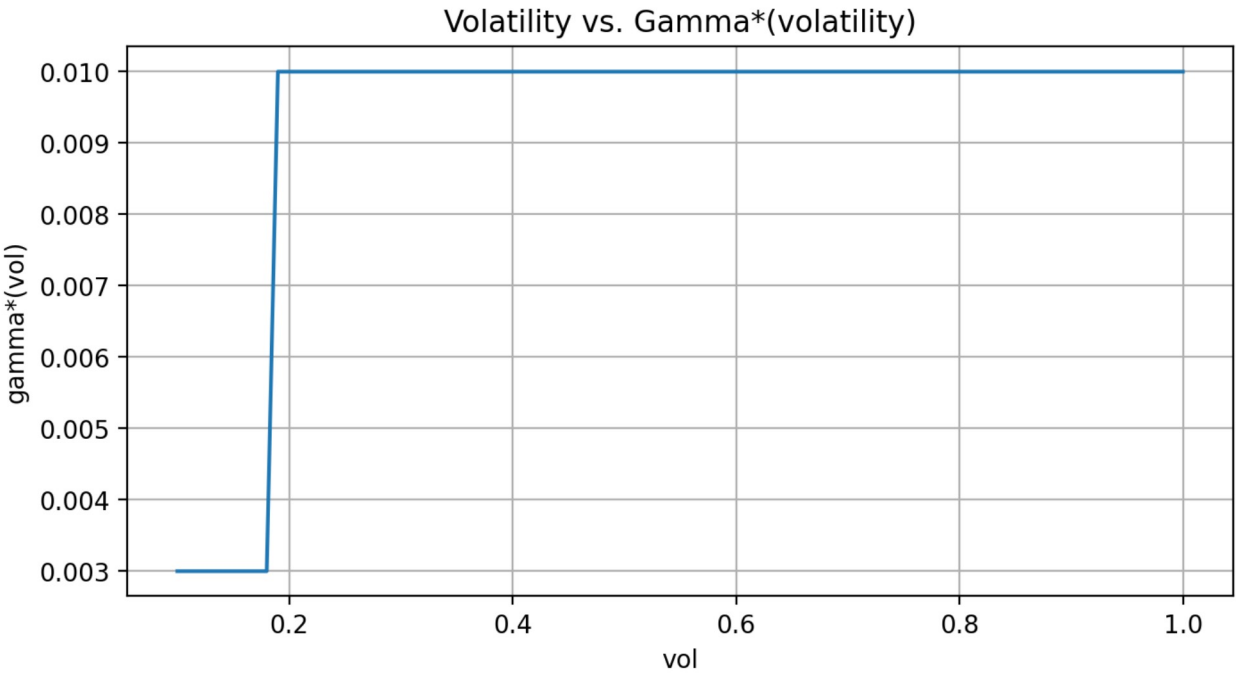
In [33]:
```
bests
```

Out[33]:

	gamma_star	E[R]_max
vol		
0.2	0.01	0.009430
0.6	0.01	0.081082
1.0	0.01	0.160690

The above table shows the best gamma*(sigma) values among the 3 options.

In [34]:
```
sigmas = np.arange(0.1, 1.01, 0.01)
gammas = [0.001, 0.003, 0.01]
x_t, y_t = 1000.0, 1000.0
dt = 1.0 / 365.0

rows = []
for s in sigmas:
    for g in gammas:
        er = expected_revenue(x_t, y_t, g, s, dt)
        rows.append({'vol': s, 'gamma': g, 'E[R]': er})

df = pd.DataFrame(rows)

idx = df.groupby('vol')['E[R]'].idxmax()
opt = df.loc[idx].sort_values('vol').reset_index(drop=True)
opt = opt.rename(columns={'gamma': 'gamma_star', 'E[R]': 'E[R]_max'})

plt.figure(figsize=(8,4), dpi=200)
plt.plot(opt['vol'], opt['gamma_star'])
plt.xlabel('vol')
plt.ylabel('gamma*(vol)')
plt.title('Volatility vs. Gamma*(volatility)')
plt.grid(True)
plt.show()
```



The above shows the plot for volatility vs optimal gamma*. The plot looks like a step function and shows us that as the volatility increases, the most optimal fee rate is 0.01. I'm sure that if we had a range of gammas to test, it would be the highest one possible. My observation is that higher vol implies higher optimal fee rates. This is probably because we make money off of as many trades as possible. With each trade, we get a payout, and obviously, as the fee rate increases, the more money we make off each transaction increases. As to why the step occurs around 0.2 volatility, this is probably because it is where the cases switch from BTC being cheaper in the pool to being cheaper outside.

Part 4

1.

Answer given from my handwritten solution. Please scroll to the end of the jupyter notebook pdf.

2.

In [35]:
```
def f1(x, y):
    return x * y

def f2(x, y):
    return np.exp(x + y)


x0 = 0.0
x1 = 1.0 # x_n+1 (last x) I named x1 for brevity
y0 = 0.0
y1 = 3.0 # y_m+1 (last y)

def comp_trapz_func(f, dx, dy):
    n = int((x1 - x0) / dx)
    m = int((y1 - y0) / dy)

    res = 0.0

    for i in range(n):
        xi = x0 + i * dx
        xi1 = xi + dx
```

```
        xmid = (xi + xi1) / 2.0
        for j in range(m):
            yj = y0 + j *dy
            yj1 = yj + dy
            ymid = (yj + yj1) / 2.0

            val = (
                f(xi, yj) + f(xi, yj1) + f(xi1, yj) +
                f(xi1, yj1) + 2 * (f(xmid, yj) +
                f(xmid, yj1) + f(xi, ymid) +
                f(xi1, ymid)) + 4 * f(xmid, ymid)
            )

            res += val

    res *= (dx * dy) / 16.0
    return res

pairs = [(0.1, 0.1), (1.0, 1.0), (0.5, 1.5), (0.5, 0.5), (1.5, 0.5), (0.5, 5.0)]

f1_actual = 9.0 / 4.0
f2_actual = np.exp(4.0) - np.exp(3.0) - np.exp(1.0) + 1.0
results = []
for in_dx, in_dy in pairs:
    f1_res = comp_trapz_func(f1, in_dx, in_dy)
    f2_res = comp_trapz_func(f2, in_dx, in_dy)
    f1_err = abs(f1_actual - f1_res)
    f2_err = abs(f2_actual - f2_res)
    results.append({
        'dx': in_dx,
        'dy': in_dy,
        'f1': f1_res,
        'f1_err': f1_err,
        'f2': f2_res,
        'f2_err': f2_err,
    })

results = pd.DataFrame(results).round(4); results
```

Out[35]:

	dx	dy	f1	f1_err	f2	f2_err
0	0.1	0.1	2.25	0.00	32.8080	0.0137
1	1.0	1.0	2.25	0.00	34.1692	1.3749
2	0.5	1.5	2.25	0.00	34.4959	1.7016
3	0.5	0.5	2.25	0.00	33.1365	0.3421
4	1.5	0.5	0.00	2.25	0.0000	32.7943
5	0.5	5.0	0.00	2.25	0.0000	32.7943

The table above displays the results for the corresponding dx, dy pairs. Firstly, I selected 2 pairs whose value should return 0 to ensure I implemented the integral approximation correctly. So for (1.5, 0.5) and (0.5, 5.0) we should get a value of 0 because the step (dx or dy) selected is too big for the bounds of the integrals. Now when looking at the other results, we get perfect approximations with no errors for all the f1 calculations. I assume this is because of how simple the function is. When looking at f2, as expected, the pair with the smallest steps of 0.1 for both x and y was the most accurate. This makes sense because the smaller we choose the step, the more "resolution" we get and the more accurate our result.

In [ ]:

(a) (10 pts) Derive the swap amounts

Under both Case 1 and Case 2, use the corresponding boundary condition, and the reserve updates above, to derive the swap size:

$$\Delta x(S_{t+1}; x_t, y_t, \gamma, k), \quad \Delta y(S_{t+1}; x_t, y_t, \gamma, k)$$

such that the one-step fee revenue is:

$$R(S_{t+1}) = \mathbf{1}_{\{S_{t+1} > \frac{P_t}{1-\gamma}\}} \gamma \, \Delta y + \mathbf{1}_{\{S_{t+1} < P_t(1-\gamma)\}} \gamma \, \Delta x \, S_{t+1}.$$

where we used the notation $1_{\{\cdot\}}$ for the indicator function.

Case 1: $S_{t+1} > P_t \frac{1}{1-\gamma}$ **(BTC cheaper in the pool)**

Arbitragers will swap USDC → BTC until the resulting band after the trades moves so that it contains the outside price S_{t+1}. Updates:

$$x_{t+1} = x_t - \Delta x, \qquad y_{t+1} = y_t + (1-\gamma)\Delta y, \qquad \Delta x, \Delta y > 0,$$

with boundary condition

$$P_{t+1}\frac{1}{1-\gamma} = \frac{y_{t+1}}{x_{t+1}}\frac{1}{1-\gamma} = S_{t+1},$$

The corresponding fee revenue (USDC) is coming from the input asset y. It is thus equal to $\gamma \, \Delta y$.

3a) Case 1:

Find Δx & Δy.

We know,

$x_{t+1} y_{t+1} = x_t y_t = k \Rightarrow x_{t+1} = \frac{k}{y_{t+1}}$

$x_{t+1} = x_t - \Delta x$

$y_{t+1} = y_t + (1-\gamma)\Delta y$

$\frac{y_{t+1}}{x_{t+1}} \frac{1}{1-\gamma} = S_{t+1} \Rightarrow x_{t+1}^2 \cdot \frac{y_{t+1}}{x_{t+1}} = (1-\gamma) S_{t+1} \cdot x_{t+1}^2$

$x_{t+1} y_{t+1} = (1-\gamma) S_{t+1} x_{t+1}^2$

$\frac{k}{(1-\gamma) S_{t+1}} = x_{t+1}^2$

$x_t - \Delta x = x_{t+1} = \sqrt{\frac{k}{(1-\gamma)S_{t+1}}}$

$\Delta x = x_t - \sqrt{\frac{k}{(1-\gamma)S_{t+1}}}$

$x_{t+1} = \sqrt{\frac{k}{(1-\gamma)S_{t+1}}}$

$\frac{k}{y_{t+1}} = \sqrt{\frac{k}{(1-\gamma)S_{t+1}}}$

$k\sqrt{\frac{(1-\gamma)S_{t+1}}{k}} = y_{t+1}$

$y_t + (1-\gamma)\Delta y = k\sqrt{\frac{(1-\gamma)S_{t+1}}{k}}$

$\Delta y = \left(k\sqrt{\frac{(1-\gamma)S_{t+1}}{k}} - y_t\right)\left(\frac{1}{1-\gamma}\right)$

$\Delta y = \left(\sqrt{k} \cdot \frac{\sqrt{(1-\gamma)S_{t+1}}}{\sqrt{k}} - y_t\right)(\cdots)$

$\Delta y = \frac{\sqrt{k(1-\gamma)S_{t+1}} - y_t}{(1-\gamma)}$

(a) (10 pts) Derive the swap amounts

Under both Case 1 and Case 2, use the corresponding boundary condition, and the reserve updates above, to derive the swap size:

$$\Delta x(S_{t+1}; x_t, y_t, \gamma, k), \quad \Delta y(S_{t+1}; x_t, y_t, \gamma, k)$$

such that the one-step fee revenue is:

$$R(S_{t+1}) = \mathbf{1}_{\{S_{t+1} > \frac{P_t}{1-\gamma}\}} \gamma \, \Delta y + \mathbf{1}_{\{S_{t+1} < P_t(1-\gamma)\}} \gamma \, \Delta x \, S_{t+1}.$$

where we used the notation $1_{\{\cdot\}}$ for the indicator function.

Case 2: $S_{t+1} < P_t(1-\gamma)$ **(BTC cheaper outside)**

Arbitragers will swap BTC → USDC and the updated reserves are:

$$x_{t+1} = x_t + (1-\gamma)\Delta x, \qquad y_{t+1} = y_t - \Delta y, \qquad \Delta x, \Delta y > 0,$$

with boundary condition

$$P_{t+1}(1-\gamma) = \frac{y_{t+1}}{x_{t+1}}(1-\gamma) = S_{t+1},$$

and the fee revenue is $\gamma \Delta x$. After converting BTC fee to USDC using S_{t+1} is calculated as $\gamma \, \Delta x \, S_{t+1}$.

3a) Case 2:

We know,

$x_{t+1} y_{t+1} = x_t y_t = k$

$x_{t+1} = x_t + (1-\gamma)\Delta x$

$y_{t+1} = y_t - \Delta y$

$\frac{y_{t+1}}{x_{t+1}}(1-\gamma) = S_{t+1} \Rightarrow x_{t+1} y_{t+1} = \frac{S_{t+1}}{(1-\gamma)} x_{t+1}^2$

$\sqrt{\frac{k(1-\gamma)}{S_{t+1}}} = x_{t+1}$

$\frac{\sqrt{\frac{k(1-\gamma)}{S_{t+1}}} - x_t}{(1-\gamma)} = \Delta x$

$\frac{k}{y_{t+1}} = \sqrt{\frac{k(1-\gamma)}{S_{t+1}}}$

$y_{t+1} = k\sqrt{\frac{S_{t+1}}{k(1-\gamma)}} = \sqrt{\frac{k S_{t+1}}{(1-\gamma)}}$

$\Delta y = y_t - \sqrt{\frac{k S_{t+1}}{(1-\gamma)}}$

Part 4. (**Bonus** 10 points) Consider the following functions:

$$f_1(x, y) = xy$$
$$f_2(x, y) = e^{x+y}$$

1. Analytically solve the following integral for both f_1 and f_2

$$\int_0^1 \int_0^3 f_i(x, y) \mathrm{d}y\mathrm{d}x$$

1)

$f_1(x,y) = xy$

$$\int_0^1 \int_0^3 xy \, dy dx = \int_0^1 \left(\frac{xy^2}{2}\right)\Big|_0^3 dx = \int_0^1 \frac{9x}{2} dx = \frac{9}{2}\left(\frac{x^2}{2}\Big|_0^1\right) = \frac{9}{2}\cdot\frac{1}{2} = \frac{9}{4}$$

$\frac{x}{2}(9-0) = \frac{9x}{2}$

$f_2(x,y) = e^{x+y}$

$$\int_0^1 \int_0^3 e^{x+y} dy dx = \int_0^1 \int_0^3 e^x e^y dy \, dx = \int_0^1 e^x \left(e^y\Big|_0^3\right) dx = \int_0^1 e^x (e^3-1) dx = (e^3-1)(e^1-1) = e^4 - e^3 - e + 1$$

$e^3 - 1$