FE621 Computational Finance

Homework #1

Name: Bence Marton, Date: 2026-02-15

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:

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

In [1]:
```
# improting libraries
import yfinance as yf
import time
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy.stats as si
import math
from datetime import date, timedelta
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
def download_data(tickers, start_date, end_date, months_ahead=4):

    def third_Friday(d): # filtering for 3rd friday expirations
        base = date(d.year, d.month, 15)
        return base + timedelta((4 - base.weekday()) % 7)

    def first_day_next_month(d):
        return (d.replace(day=1) + timedelta(days=32)).replace(day=1)

    def collecting_data(start, k): # downloading data
        current = start.date() if hasattr(start, "date") else start
        res = []
        while len(res) < k:
            f = third_Friday(current)
            if f >= current:
                res.append(f.strftime("%Y-%m-%d"))
            current = first_day_next_month(current)
        return res

    today = pd.to_datetime("today")
    exp_monthly = collecting_data(today, months_ahead)

    vix_exp = [(pd.to_datetime(e) - timedelta(days=2)).strftime("%Y-%m-%d") for e in exp_monthly]

    stocks_all = [] # empty stock df
    options_all = [] # empty option df

    for ticker in tickers:
        tkr = yf.Ticker(ticker)
        # getting stock data
        stock_data_download = tkr.history(start=start_date, end=end_date)
        stock_tmp = stock_data_download.copy()
        stock_tmp["ticker"] = ticker
        stock_tmp["date"] = stock_tmp.index
        stocks_all.append(stock_tmp.reset_index(drop=True))

        available_exp = set(getattr(tkr, "options", []) or [])
        target_exp = vix_exp if ticker == "^VIX" else exp_monthly
        exp_fetch = [e for e in target_exp if e in available_exp]

        for e in exp_fetch:
            ch = tkr.option_chain(e)
            # getting option data
            call_options = ch.calls.dropna()
            call_options = call_options.loc[call_options["impliedVolatility"] > 0].copy()
            call_options["exp"] = e
            call_options["type"] = "call"
            call_options["ticker"] = ticker
            options_all.append(call_options)
            put_options = ch.puts.dropna()
            put_options = put_options.loc[put_options["impliedVolatility"] > 0].copy()

            put_options["exp"] = e
            put_options["type"] = "put"
            put_options["ticker"] = ticker
            options_all.append(put_options)

    stocks_df = pd.concat(stocks_all, ignore_index=True) if stocks_all else pd.DataFrame()
    options_df = pd.concat(options_all, ignore_index=True) if options_all else pd.DataFrame()

    return stocks_df, options_df
```

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

- TSLA
- SPY
- ^VIX

In [3]:
```
tickers = ['TSLA', 'SPY', '^VIX']
start_date = '2026-02-12'
end_date = '2026-02-14'
downloaded_data = download_data(tickers,start_date, end_date) # downloading and saving the data
```

All option data downloaded on the 14th of Feb from yfinance

In [4]:
```
option_data = downloaded_data[1] # options data only
option_data
```

Out[4]:

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	impliedVolatility	inTheMoney	contractSize	currency	exp	type	ticker
0	TSLA260220C00100000	2026-02-12 20:10:06+00:00	100.0	315.58	316.50	318.65	0.000000	0.000000	4.0	3866	4.210942	True	REGULAR	USD	2026-02-20	call	TSLA
1	TSLA260220C00110000	2025-11-07 14:45:01+00:00	110.0	321.40	321.85	324.05	0.000000	0.000000	1.0	22	9.394047	True	REGULAR	USD	2026-02-20	call	TSLA
2	TSLA260220C00120000	2026-01-16 17:08:41+00:00	120.0	319.63	295.40	299.70	0.000000	0.000000	90.0	141	3.609376	True	REGULAR	USD	2026-02-20	call	TSLA
3	TSLA260220C00130000	2026-01-16 16:40:36+00:00	130.0	309.87	285.40	289.70	0.000000	0.000000	5.0	43	3.386720	True	REGULAR	USD	2026-02-20	call	TSLA
4	TSLA260220C00140000	2026-01-16 20:53:02+00:00	140.0	299.78	275.45	279.70	0.000000	0.000000	6.0	29	3.250002	True	REGULAR	USD	2026-02-20	call	TSLA
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2872	VIX260415P00090000	2026-02-04 16:09:05+00:00	90.0	69.15	68.30	68.95	0.000000	0.000000	2.0	14	0.000010	True	REGULAR	USD	2026-04-15	put	^VIX
2873	VIX260415P00100000	2026-02-13 16:24:16+00:00	100.0	78.77	0.00	0.00	-0.400002	-0.505244	3.0	33	0.000010	True	REGULAR	USD	2026-04-15	put	^VIX
2874	VIX260415P00110000	2026-01-05 19:38:52+00:00	110.0	88.90	88.90	89.60	0.000000	0.000000	2.0	2	1.929688	True	REGULAR	USD	2026-04-15	put	^VIX
2875	VIX260415P00180000	2026-01-05 15:35:25+00:00	180.0	158.10	158.30	159.05	0.000000	0.000000	48.0	0	0.000010	True	REGULAR	USD	2026-04-15	put	^VIX
2876	VIX260415P00200000	2026-02-13 16:24:16+00:00	200.0	177.99	177.30	178.30	-0.189987	-0.106627	3.0	362	0.000010	True	REGULAR	USD	2026-04-15	put	^VIX

2877 rows × 17 columns

Stock Data

In [5]:
```
DATA1 = downloaded_data[0][downloaded_data[0]["date"].dt.date == pd.to_datetime("2026-02-12").date()]
DATA2 = downloaded_data[0][downloaded_data[0]["date"].dt.date == pd.to_datetime("2026-02-13").date()]
```

In [6]:
```
dropping_columns = ["Dividends", "Stock Splits", "Capital Gains"]
DATA1 = (DATA1.drop(columns=dropping_columns, errors="ignore").loc[:, ["ticker"] + [c for c in DATA1.columns if c not in ["ticker"] + dropping_columns]])
DATA2 = (DATA2.drop(columns=dropping_columns, errors="ignore").loc[:, ["ticker"] + [c for c in DATA2.columns if c not in ["ticker"] + dropping_columns]])
```

Stock data for 12th of Feb

In [7]:
```
DATA1
```

Out[7]:

	ticker	Open	High	Low	Close	Volume	date
0	TSLA	430.299988	436.230011	414.000000	417.070007	61933400	2026-02-12 00:00:00-05:00
2	SPY	694.239990	695.349976	680.369995	681.270020	118829000	2026-02-12 00:00:00-05:00
4	^VIX	17.440001	21.209999	17.080000	20.820000	0	2026-02-12 00:00:00-05:00

Stock data for 13th of Feb

In [8]:
```
DATA2
```

Out[8]:

	ticker	Open	High	Low	Close	Volume	date
1	TSLA	414.309998	424.059998	410.880005	417.440002	51351200	2026-02-13 00:00:00-05:00
3	SPY	681.690002	686.280029	677.520020	681.750000	96150400	2026-02-13 00:00:00-05:00
5	^VIX	21.480000	22.400000	18.920000	20.600000	0	2026-02-13 00:00:00-05:00

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

- TSLA is company, producing electric cars and batteries, it`s stock represents partial ownership in the company.
- SPY is an ETF which tracks the S&P 500 index and so it`s consituten. The purpose here is for investors to be able to trade or invest in the S&P index in a very cheap way.
- ^VIX is a volatility index designed to tract the volatility of S&P 500, it measures expected volatility of the S&P 500 over the next 30 days using S&P 500 index option prices

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 data for the underlying asset as above in DATA1 and DATA2

• The short-term interest rate which may be obtained here: http://www.federalreserve.gov/releases/H15/Current/.

I downloaded from yfinance too, to be consitent. ^IRX is the ticker for the 3 months treasury bill.

In [9]:
```
risk_free_rate = yf.download("^IRX",period='1d')["Close"] #gettin data from yfinance
risk_free_rate
```

```
[*********************100%***********************]  1 of 1 completed
YF.download() has changed argument auto_adjust default to True
```

Out[9]:

Ticker	^IRX
Date	
2026-02-13	3.593

In [10]:
```
t_bill = risk_free_rate["^IRX"][0]/100 # converting to decimals
```

• Time to Maturity

In [11]:
```
option_data['exp'] = pd.to_datetime(option_data['exp'])
today = pd.Timestamp.today().normalize()
option_data['Time_to_Maturity'] = (option_data['exp'] - today).dt.days/365 # calculting Time_to_Maturity
option_data
```

Out[11]:

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	impliedVolatility	inTheMoney	contractSize	currency	exp	type	ticker	Time_to_Maturity
0	TSLA260220C00100000	2026-02-12 20:10:06+00:00	100.0	315.58	316.50	318.65	0.000000	0.000000	4.0	3866	4.210942	True	REGULAR	USD	2026-02-20	call	TSLA	0.013699
1	TSLA260220C00110000	2025-11-07 14:45:01+00:00	110.0	321.40	321.85	324.05	0.000000	0.000000	1.0	22	9.394047	True	REGULAR	USD	2026-02-20	call	TSLA	0.013699
2	TSLA260220C00120000	2026-01-16 17:08:41+00:00	120.0	319.63	295.40	299.70	0.000000	0.000000	90.0	141	3.609376	True	REGULAR	USD	2026-02-20	call	TSLA	0.013699
3	TSLA260220C00130000	2026-01-16 16:40:36+00:00	130.0	309.87	285.40	289.70	0.000000	0.000000	5.0	43	3.386720	True	REGULAR	USD	2026-02-20	call	TSLA	0.013699
4	TSLA260220C00140000	2026-01-16 20:53:02+00:00	140.0	299.78	275.45	279.70	0.000000	0.000000	6.0	29	3.250002	True	REGULAR	USD	2026-02-20	call	TSLA	0.013699
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2872	VIX260415P00090000	2026-02-04 16:09:05+00:00	90.0	69.15	68.30	68.95	0.000000	0.000000	2.0	14	0.000010	True	REGULAR	USD	2026-04-15	put	^VIX	0.161644
2873	VIX260415P00100000	2026-02-13 16:24:16+00:00	100.0	78.77	0.00	0.00	-0.400002	-0.505244	3.0	33	0.000010	True	REGULAR	USD	2026-04-15	put	^VIX	0.161644
2874	VIX260415P00110000	2026-01-05 19:38:52+00:00	110.0	88.90	88.90	89.60	0.000000	0.000000	2.0	2	1.929688	True	REGULAR	USD	2026-04-15	put	^VIX	0.161644
2875	VIX260415P00180000	2026-01-05 15:35:25+00:00	180.0	158.10	158.30	159.05	0.000000	0.000000	48.0	0	0.000010	True	REGULAR	USD	2026-04-15	put	^VIX	0.161644
2876	VIX260415P00200000	2026-02-13 16:24:16+00:00	200.0	177.99	177.30	178.30	-0.189987	-0.106627	3.0	362	0.000010	True	REGULAR	USD	2026-04-15	put	^VIX	0.161644

2877 rows × 18 columns

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 S0, volatility , 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).

In [12]:

```python
class BlackScholesModel:
    def __init__(self, S, K, T, r, sigma):
        self.S = S        # Underlying asset price
        self.K = K        # Option strike price
        self.T = T        # Time to expiration in years
        self.r = r        # Risk-free interest rate
        self.sigma = sigma  # Volatility of the underlying asset

    def d1(self):
        return (np.log(self.S / self.K) + (self.r + 0.5 * self.sigma ** 2) * self.T) / (self.sigma * np.sqrt(self.T))

    def d2(self):
        return self.d1() - self.sigma * np.sqrt(self.T)

    def call_option_price(self): #calculating call price
        return (self.S * si.norm.cdf(self.d1(), 0.0, 1.0) - self.K * np.exp(-self.r * self.T) * si.norm.cdf(self.d2(), 0.0, 1.0))

    def put_option_price(self): #calculating put price
        return (self.K * np.exp(-self.r * self.T) * si.norm.cdf(-self.d2(), 0.0, 1.0) - self.S * si.norm.cdf(-self.d1(), 0.0, 1.0))

    def delta_call(self):#calculating delta call
        return si.norm.cdf(self.d1(), 0.0, 1.0)
    def delta_put(self):#calculating delta put
        return -si.norm.cdf(-self.d1(), 0.0, 1.0)
    def gamma(self):#calculating gamma
```

```
        return si.norm.pdf(self.d1(), 0.0, 1.0) / (self.S * self.sigma * np.sqrt(self.T))
    def vega(self):#calculating vega
        return self.S * si.norm.pdf(self.d1(), 0.0, 1.0) * np.sqrt(self.T)
```

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.

In [13]:
```
data_day = pd.to_datetime("2026-02-12") #filter for the 12th of feb
r = t_bill # risk free rate
tolerance_level = 1e-6 # tolerance level as given
option_data["mid"] = (option_data["bid"] + option_data["ask"]) / 2 # mid price
Time_to_Maturity = option_data['Time_to_Maturity'] # Time_to_Maturity
```

In [14]:
```
def bisection_method(S=100, K=50, T=1, r=t_bill, x1=0, x2=1, option_price=1, option_type="call",
                     tolerance_level=10e-6,max_iter=200):

    #helper function to determine the implied volatility that makes the Black-Scholes price equal
    #to the current market price.
    def price_error(sigma):
        model = BlackScholesModel(S, K, T, r, sigma)
        if option_type.lower() == "call":
            return model.call_option_price() - option_price
        elif option_type.lower() == "put":
            return model.put_option_price() - option_price
        else:
            raise ValueError("option_type must be 'call' or 'put'")
    #function at the endpoints
    f1 = price_error(x1)
    f2 = price_error(x2)

    #Bisection needs the root to be bracketed

    if f1 * f2 >= 0:
        return None

    for i in range(max_iter):
        # midpoint interval
        x3 = 0.5 * (x1 + x2)
        f3 = price_error(x3) # stop if function value is close to zero
        if abs(f3) < tolerance_level or (x2 - x1) < tolerance_level:
            return x3
        if f1 * f3 < 0:
            x2, f2 = x3, f3
        else:
            x1, f1 = x3, f3
    return 0.5 * (x1 + x2) #return midpoint
```

In [15]:
```
mny_lo, mny_hi = 0.90, 1.10   # the bands to check between in-the-money and out-of-the-money options

spot = (DATA1.loc[pd.to_datetime(DATA1["date"]).dt.tz_convert(None).dt.normalize() == pd.Timestamp(data_day),
                  ["ticker", "Close"]].set_index("ticker")["Close"].to_dict()) #DATA1 Spot price

# filter for VIX
bisection_data = option_data.copy()
bisection_data = bisection_data.loc[bisection_data["ticker"] != "^VIX"].copy()

bisection_data = bisection_data.loc[bisection_data["bid"].notna() & bisection_data["ask"].notna() &
                                    (bisection_data["volume"].fillna(0) > 0) & (bisection_data["bid"] > 0) &
                                    (bisection_data["ask"] > 0)].copy()

# mid price
bisection_data["mid"] = option_data["mid"]
```

```python
results = []

for tkr, S0 in spot.items():
    if tkr == "^VIX":
        continue

    sub = bisection_data.loc[bisection_data["ticker"] == tkr].copy()
    if sub.empty:
        continue

    # nearest expiration
    nearest_exp = sub["exp"].min()
    sub = sub.loc[sub["exp"] == nearest_exp].copy()

    # strike closest to S0 = ATM
    sub["abs_diff"] = (sub["strike"] - S0).abs()

    # select ATM call and ATM put separately
    atm_call = sub.loc[sub["type"].str.lower() == "call"].sort_values("abs_diff").head(1)
    atm_put  = sub.loc[sub["type"].str.lower() == "put"].sort_values("abs_diff").head(1)

    atm_call_iv = np.nan
    atm_put_iv  = np.nan

    if len(atm_call) == 1:
        row = atm_call.iloc[0]
        atm_call_iv = bisection_method(S=S0, K=row["strike"], T=row["Time_to_Maturity"], r=r,x1=1e-6, x2=2,
                                       option_price=row["mid"], option_type="call",tolerance_level=tolerance_level)

    if len(atm_put) == 1:
        row = atm_put.iloc[0]
        atm_put_iv = bisection_method(S=S0, K=row["strike"], T=row["Time_to_Maturity"], r=r,x1=1e-6, x2=2,
                                      option_price=row["mid"], option_type="put",tolerance_level=tolerance_level)

    # between ITM and OTM average
    sub["mny"] = S0 / sub["strike"]
    band = sub.loc[(sub["mny"] >= mny_lo) & (sub["mny"] <= mny_hi)].copy()

    if not band.empty:
        band["iv"] = band.apply(lambda row: bisection_method(S=S0, K=row["strike"], T=row["Time_to_Maturity"], r=r,
                                                             x1=1e-6, x2=2,
                                                             option_price=row["mid"], option_type=row["type"],
                                                             tolerance_level=tolerance_level),axis=1)
        avg_iv = float(band["iv"].dropna().mean()) if band["iv"].notna().any() else np.nan
    else:
        avg_iv = np.nan

    results.append({
        "ticker": tkr,
        "DATA1_day": str(pd.Timestamp(data_day).date()),
        "S0": float(S0),
        "nearest_exp": str(nearest_exp.date()),
        "ATM_call_iv": atm_call_iv,
        "ATM_put_iv": atm_put_iv,
        "avg_iv_ITM_OTM_band": avg_iv,
        "band": f"{mny_lo}-{mny_hi}"
    })
#Creating datafame
implied_volatility_bisection = pd.DataFrame(results)
implied_volatility_bisection
```

Out[15]:

	ticker	DATA1_day	S0	nearest_exp	ATM_call_iv	ATM_put_iv	avg_iv_ITM_OTM_band	band
0	TSLA	2026-02-12	417.070007	2026-02-20	0.467827	0.443456	0.468279	0.9-1.1
1	SPY	2026-02-12	681.270020	2026-02-20	0.214124	0.195086	0.252353	0.9-1.1

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 . 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.

In [16]:
```python
def newton_method_vol(S0, K, T, r, opt_price, option_type="call", tolerance=1e-8, sigma_0=0.20, max_iter=100):

    sigma = float(sigma_0)
    sigma = max(sigma, 1e-8)

    for i in range(max_iter):
        model = BlackScholesModel(S=S0, K=K, T=T, r=r, sigma=sigma)

        if option_type.lower() == "call":
            price = model.call_option_price()
        elif option_type.lower() == "put":
            price = model.put_option_price()

        vega = model.vega()
        diff = price - opt_price

        # stop if price is close
        if abs(diff) < tolerance:
            return sigma

        # avoid division by zero
        if abs(vega) < 1e-12:
            return np.nan

        # Creates New sigma
        sigma_new = sigma - diff / vega

        # keep sigma in positive range
        sigma_new = float(np.clip(sigma_new, 1e-8, 5.0))

        # stop if sigma not changing
        if abs(sigma_new - sigma) < tolerance:
            return sigma_new

        sigma = sigma_new

    return sigma
```

In [17]:
```python
# remove VIX
newton_data = option_data[option_data["ticker"] != "^VIX"].copy()

# valid quotes filtering
newton_data = newton_data[(newton_data["bid"].notna()) & (newton_data["ask"].notna()) & (newton_data["volume"] > 0) &
                          (newton_data["bid"] > 0) & (newton_data["ask"] > 0)].copy()
# calculating mid price
newton_data["mid"] = option_data["mid"]
```

In [18]:
```python
results = []

for ticker in DATA1["ticker"].unique():

    if ticker == "^VIX":
        continue
    S0 = DATA1.loc[DATA1["ticker"] == ticker, "Close"].iloc[0]
    sub = newton_data[newton_data["ticker"] == ticker].copy()

    # nearest expiration
    nearest_exp = sub["exp"].min()
    sub = sub[sub["exp"] == nearest_exp]

    # ATM strike
    sub["dist"] = abs(sub["strike"] - S0)
    atm = sub.loc[sub["dist"].idxmin()]

    #Newton timing
    start = time.perf_counter()
    iv_newton = newton_method_vol(S0, atm["strike"], atm["Time_to_Maturity"], r, atm["mid"])
    newton_time = time.perf_counter() - start

    #Bisection timing
    start = time.perf_counter()
    iv_bisect = bisection_method(S=S0, K=atm["strike"], T=atm["Time_to_Maturity"], r=r,
                             x1=1e-6, x2=2, option_price=atm["mid"],
                             option_type=atm["type"], tolerance_level=tolerance_level)
    bisect_time = time.perf_counter() - start

    #Average IVs
    sub["mny"] = S0 / sub["strike"]
    band = sub[(sub["mny"] >= 0.9) & (sub["mny"] <= 1.1)].copy()

    band["iv_newton"] = band.apply(
        lambda row: newton_method_vol(S0, row["strike"], row["Time_to_Maturity"], r, row["mid"]),axis=1)

    avg_iv = band["iv_newton"].mean()

    results.append([ticker, iv_newton, iv_bisect, newton_time, bisect_time, avg_iv])
```

In [19]:
```
#creating dataframe
iv_newton = pd.DataFrame(results, columns=[
    "Ticker",
    "ATM_IV_Newton",
    "ATM_IV_Bisection",
    "Newton_Time",
    "Bisection_Time",
    "Average_IV_0.9_1.1"
])

iv_newton
```

Out[19]:

	Ticker	ATM_IV_Newton	ATM_IV_Bisection	Newton_Time	Bisection_Time	Average_IV_0.9_1.1
0	TSLA	0.467827	0.467827	0.001186	0.004346	0.923353
1	SPY	0.214124	0.214124	0.000913	0.004066	0.519920

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 diference 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.

In [20]:
```
# filter out VIX
IV_values = option_data[option_data["ticker"] != "^VIX"].copy()

# filter valid quotes only
IV_values = IV_values[(IV_values["bid"].notna()) & (IV_values["ask"].notna()) & (IV_values["volume"] > 0) &
                      (IV_values["bid"] > 0) & (IV_values["ask"] > 0)].copy()
IV_values["mid"] = option_data["mid"] # mid price
IV_values = IV_values[IV_values["Time_to_Maturity"] > 0] # only positive maturity
spot_map = dict(zip(DATA1["ticker"], DATA1["Close"])) # Map spot prices
```

```
IV_values["S0"] = IV_values["ticker"].map(spot_map) # Map spot prices
IV_values = IV_values.dropna(subset=["S0"]).copy() #drop NAs
```

In [21]:

```
# Newton IV
IV_values["IV_newton"] = IV_values.apply(lambda row: newton_method_vol(row["S0"], row["strike"], row["Time_to_Maturity"],
                                                     r, row["mid"],option_type=row["type"]),axis=1)

# Bisection IV
IV_values["IV_bisection"] = IV_values.apply(lambda row: bisection_method(S=row["S0"], K=row["strike"], T=row["Time_to_Maturity"],
                                                      r=r,x1=1e-6, x2=2,
                                                      option_price=row["mid"], option_type=row["type"],
                                                      tolerance_level=1e-6),axis=1)
#creating table
IV_table = (IV_values.groupby(["ticker", "exp", "type"], as_index=False).agg(IV_newton=("IV_newton", "mean"),
                                                     IV_bisection=("IV_bisection", "mean"),
                                                     n=("IV_newton", "count")).sort_values(["ticker", "exp", "type"]))

IV_table
```

Out[21]:

	ticker	exp	type	IV_newton	IV_bisection	n
0	SPY	2026-02-20	call	0.288466	0.514352	115
1	SPY	2026-02-20	put	0.318873	0.382856	112
2	SPY	2026-03-20	call	0.252317	0.403981	152
3	SPY	2026-03-20	put	0.262511	0.369138	161
4	SPY	2026-04-17	call	0.173620	0.173620	150
5	SPY	2026-04-17	put	0.214859	0.311265	110
6	SPY	2026-05-15	call	0.183607	0.185031	109
7	SPY	2026-05-15	put	0.216649	0.252615	106
8	TSLA	2026-02-20	call	0.530358	0.742207	52
9	TSLA	2026-02-20	put	0.509003	0.806496	42
10	TSLA	2026-03-20	call	0.491896	0.655246	75
11	TSLA	2026-03-20	put	0.521123	0.691578	69
12	TSLA	2026-04-17	call	0.429007	0.603035	49
13	TSLA	2026-04-17	put	0.440379	0.661335	43
14	TSLA	2026-05-15	call	0.495034	0.682270	22
15	TSLA	2026-05-15	put	0.488529	0.723595	22

In [22]:

```
IV_values["mny"] = IV_values["S0"] / IV_values["strike"] #ratio of stock price to strike
ATM_band = IV_values[(IV_values["mny"] >= 0.9) & (IV_values["mny"] <= 1.1)] #filter for options in our band

#volatility level around the current stock price
avg_vol_table = (ATM_band.groupby("ticker")["IV_bisection"].mean().reset_index(name="Average_ATM_IV"))
avg_vol_table
```

Out[22]:

	ticker	Average_ATM_IV
0	SPY	0.196391
1	TSLA	0.462187

- Here I included both IVs calculated using the Newton and Bisection methods. It's clear that for near options the Bisection method is giving us much higher, almost double IVs compared to the Newton method. For far expiries, especially for calls, they are identical.
- Using bisection method TSLA IVs are more then double the SPY IVs. TSLA has 46% avg IV while SPY has 20% avg IV, which is not surprising given TSLA is a growth stock.
- VIX was at 20.6 on Friday, so it's fairly in line with SPY, which is realistic as it measures the volatility of the S&P 500.
- In terms of maturity, as we can see, IV declines as maturity increases.
- When the options become in the money, puts seem to have higher IVs than calls, especially for TSLA, likely investors pay more for downside protection.

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.

In [23]:
```python
# Creating put call parity table
Put_Call_table = IV_values.copy()
Put_Call_table["S0"] = Put_Call_table["ticker"].map(spot_map)
Put_Call_table = Put_Call_table.dropna(subset=["S0"])
calls = Put_Call_table[Put_Call_table["type"] == "call"].copy()
puts  = Put_Call_table[Put_Call_table["type"] == "put"].copy()
parity = pd.merge(calls,puts,on=["ticker","strike","exp"],suffixes=("_C","_P"))

parity["discK"] = parity["strike"] * np.exp(-r * parity["Time_to_Maturity_P"])

# parity prices
parity["call_from_put"] = parity["mid_P"] + parity["S0_C"] - parity["discK"]
parity["put_from_call"] = parity["mid_C"] - parity["S0_C"] + parity["discK"]

# Call price check
parity["call_inside_spread"] = ((parity["call_from_put"] >= parity["bid_C"]) & (parity["call_from_put"] <= parity["ask_C"]))

# Put price check
parity["put_inside_spread"] = ((parity["put_from_call"] >= parity["bid_P"]) & (parity["put_from_call"] <= parity["ask_P"]))
# Creating tables
parity_table = parity[["ticker","strike","exp","call_from_put","bid_C","ask_C","call_inside_spread",
                       "put_from_call","bid_P","ask_P","put_inside_spread"]]

parity_table.head(20)
```

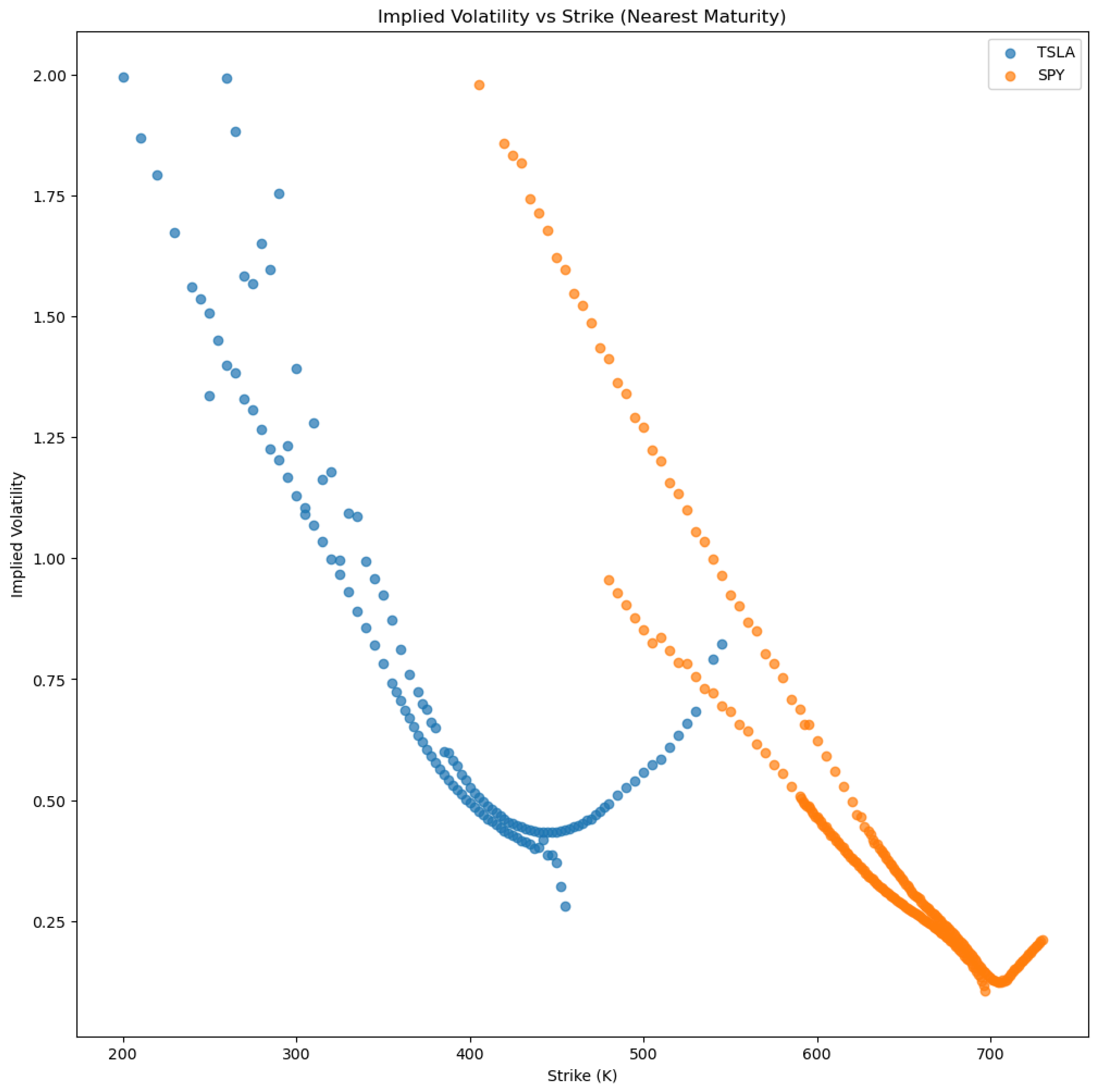
Out[23]:

	ticker	strike	exp	call_from_put	bid_C	ask_C	call_inside_spread	put_from_call	bid_P	ask_P	put_inside_spread
0	TSLA	200.0	2026-02-20	217.183421	216.55	218.75	True	0.481579	0.01	0.02	False
1	TSLA	210.0	2026-02-20	207.188342	205.50	209.75	True	0.451658	0.01	0.02	False
2	TSLA	220.0	2026-02-20	197.198263	195.55	199.75	True	0.471737	0.01	0.03	False
3	TSLA	230.0	2026-02-20	187.203184	185.60	189.75	True	0.491816	0.01	0.03	False
4	TSLA	240.0	2026-02-20	177.208104	175.75	179.75	True	0.561896	0.01	0.03	False
5	TSLA	245.0	2026-02-20	172.215565	170.55	174.75	True	0.459435	0.02	0.03	False
6	TSLA	250.0	2026-02-20	167.223025	165.70	168.70	True	0.006975	0.02	0.04	False
7	TSLA	255.0	2026-02-20	162.225485	160.65	164.75	True	0.504515	0.02	0.04	False
8	TSLA	260.0	2026-02-20	157.227946	155.85	159.75	True	0.602054	0.02	0.04	False
9	TSLA	265.0	2026-02-20	152.240406	150.70	154.75	True	0.524594	0.03	0.05	False
10	TSLA	270.0	2026-02-20	147.242866	146.55	148.25	True	0.197134	0.03	0.05	False
11	TSLA	275.0	2026-02-20	142.255327	140.80	144.10	True	0.244673	0.04	0.06	False
12	TSLA	280.0	2026-02-20	137.262787	136.70	138.65	True	0.467213	0.05	0.06	False
13	TSLA	285.0	2026-02-20	132.270247	130.90	134.50	True	0.489753	0.05	0.07	False
14	TSLA	290.0	2026-02-20	127.287708	126.80	129.80	True	1.087292	0.07	0.08	False
15	TSLA	295.0	2026-02-20	122.300168	120.85	123.85	True	0.134832	0.08	0.09	False
16	TSLA	300.0	2026-02-20	117.312629	116.80	118.55	True	0.457371	0.09	0.10	False
17	TSLA	305.0	2026-02-20	112.335089	110.90	113.75	True	0.104911	0.11	0.12	False
18	TSLA	310.0	2026-02-20	107.352549	106.85	108.55	True	0.477451	0.12	0.14	False
19	TSLA	315.0	2026-02-20	102.375010	101.75	103.40	True	0.349990	0.14	0.16	False

- For most strikes, put–call parity holds, so no arbitrage opportunity exists for calls.
- Differences occur only for deep in-the-money and out-of-the-money options; this is most probably due to low liquidity and wide spreads. Illiquid options break parity most of the time.

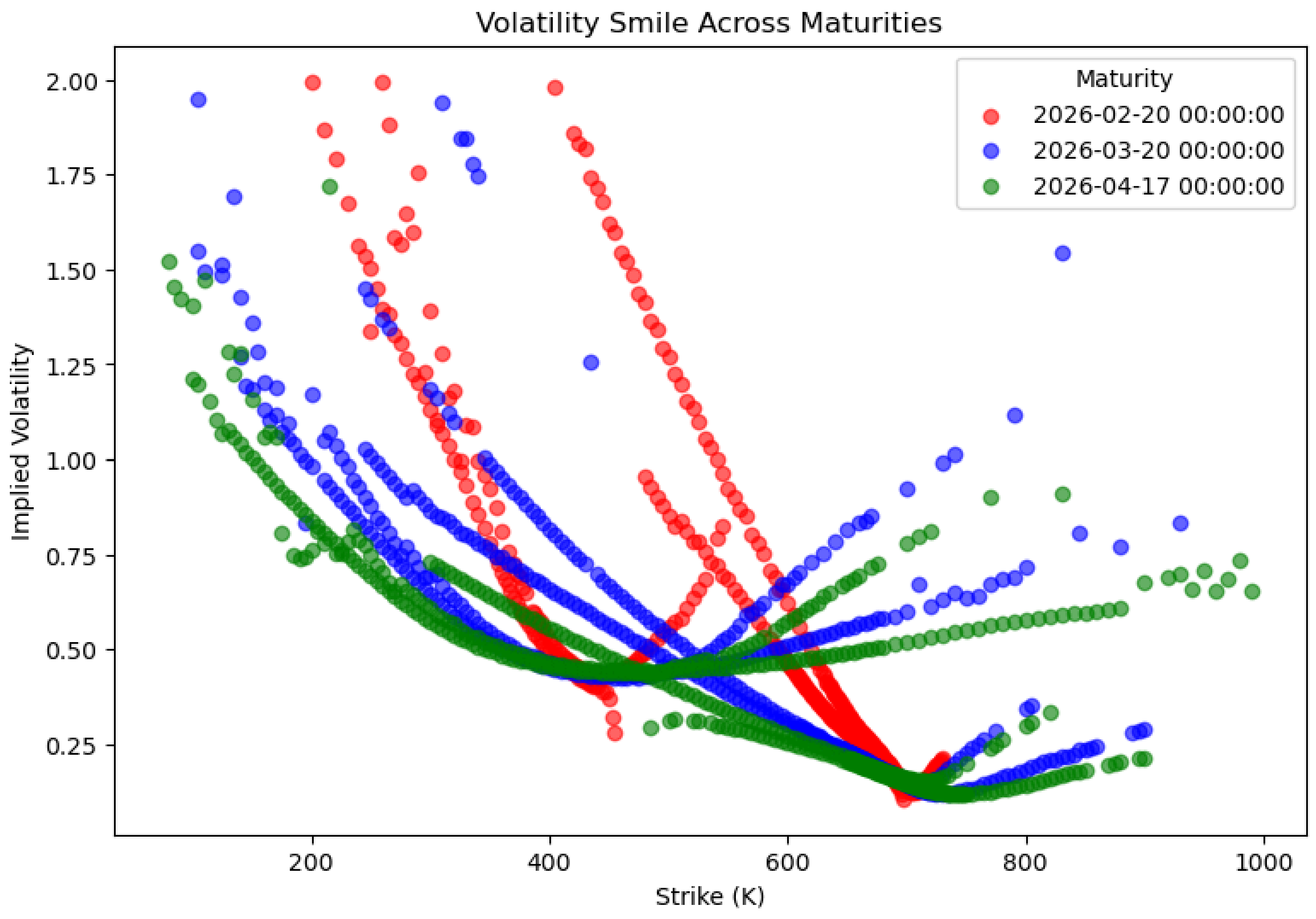
10. Consider the implied volatility values obtained in the previous parts. Create a 2 dimensional plot of implied volatilities versus strike K for the closest to maturity options. What do you observe? Plot all implied volatilities for the three different maturities on the same plot, where you use a different color for each maturity. In total there should be 3 sets of points plotted with different color.

In [24]:
```python
opt_plot = IV_values.copy()   # DATA1 options containing implied volatilities

closest_exp = opt_plot["exp"].min() #shortest maturity selection
near = opt_plot[opt_plot["exp"] == closest_exp]
plt.figure(figsize=(12,12)) # Creating the plot

#scatters to show the volatility smile
for ticker in near["ticker"].unique():
    sub = near[near["ticker"] == ticker]
    plt.scatter(sub["strike"], sub["IV_bisection"], label=ticker, alpha=0.7)
# Label axes and title
plt.xlabel("Strike (K)")
plt.ylabel("Implied Volatility")
plt.title("Implied Volatility vs Strike (Nearest Maturity)")
plt.legend()
plt.show()
```



- The volatility smile is clearly in present for SPY and TSLA, when using the bisection method.
- IV is lowest near the at-the-money strike and increases as strikes move deeper in- or out-of-the-money.
- TSLA has much steeper smile, this is due to strong skew and demand fro riks protection.

In [25]:
```python
plt.figure(figsize=(9,6)) # Creating the plot

expirations = sorted(opt_plot["exp"].unique())[:3] # selecting 3 expirations
colors = ["red","blue","green"]

#scatters to show the volatility smile
for exp, color in zip(expirations, colors):
    sub = opt_plot[opt_plot["exp"] == exp]
    plt.scatter(sub["strike"], sub["IV_bisection"], color=color, label=str(exp), alpha=0.6)
# Label axes and title
plt.xlabel("Strike (K)")
plt.ylabel("Implied Volatility")
plt.title("Volatility Smile Across Maturities")
plt.legend(title="Maturity")
plt.show()
```



- The steepest curves are present for the red dotted options, which are near expiry, while the opposite holds for the green longer-maturity options.
- Near at-the-money options have low IV, while it increases for lower and higher strike options.

11. (Greeks) Calculate the derivatives of the call option price with respect to S (Delta), and (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.

In [26]:
```python
calls = option_data.copy() #creating dataframe for call options
calls = calls[(calls["type"] == "call") & (calls["ticker"] != "^VIX")].copy()

# map the spot price S0 from DATA1
spot_map = dict(zip(DATA1["ticker"], DATA1["Close"]))
calls["S0"] = calls["ticker"].map(spot_map) #Map spot prices
calls = calls.dropna(subset=["S0"])
```

```python
# time to maturity
calls["Time_to_Maturity"] = (pd.to_datetime(calls["exp"]) - data_day).dt.days / 365
calls = calls[calls["Time_to_Maturity"] > 0].copy()
# renaming for vol
calls["sigma"] = calls["impliedVolatility"]


h_frac = 0.01      # 1% for delta and gamma approximation
v_step = 1e-4      # step for vega

rows = []

for i, row in calls.iterrows():
    S = float(row["S0"])
    K = float(row["strike"])
    T = float(row["Time_to_Maturity"])
    sig = float(row["sigma"])

    if sig <= 0 or T <= 0 or S <= 0 or K <= 0:
        continue

    # calculating greeks
    m = BlackScholesModel(S=S, K=K, T=T, r=r, sigma=sig)
    C0 = m.call_option_price()
    delta_a = m.delta_call()
    gamma_a = m.gamma()
    vega_a  = m.vega()

    # steps
    h = h_frac * S
    vs = v_step

    # finite method
    Cp = BlackScholesModel(S=S + h, K=K, T=T, r=r, sigma=sig).call_option_price()
    Cm = BlackScholesModel(S=S - h, K=K, T=T, r=r, sigma=sig).call_option_price()

    delta_fd = (Cp - Cm) / (2 * h)
    gamma_fd = (Cp - 2 * C0 + Cm) / (h ** 2)

    # finite method for sigma
    Cvp = BlackScholesModel(S=S, K=K, T=T, r=r, sigma=sig + vs).call_option_price()
    Cvm = BlackScholesModel(S=S, K=K, T=T, r=r, sigma=max(sig - vs, 1e-8)).call_option_price()
    vega_fd = (Cvp - Cvm) / (2 * vs)

    rows.append({
        "ticker": row["ticker"],
        "exp": row["exp"],
        "K": K,
        "S0": S,
        "Time_to_Maturity": T,
        "sigma": sig,

        "Delta_BS": delta_a,
        "Delta_FD": delta_fd,
        "Delta_diff": delta_fd - delta_a,

        "Gamma_BS": gamma_a,
        "Gamma_FD": gamma_fd,
        "Gamma_diff": gamma_fd - gamma_a,

        "Vega_BS": vega_a,
        "Vega_FD": vega_fd,
        "Vega_diff": vega_fd - vega_a,
    })
```

```python
greeks_table = pd.DataFrame(rows)
greeks_table.head(20)
```

Out[26]:

	ticker	exp	K	S0	Time_to_Maturity	sigma	Delta_BS	Delta_FD	Delta_diff	Gamma_BS	Gamma_FD	Gamma_diff	Vega_BS	Vega_FD	Vega_diff
0	TSLA	2026-02-20	100.0	417.070007	0.021918	4.210942	0.995389	0.995387	-0.000002	0.000052	0.000052	1.267764e-08	0.830631	0.830631	8.504815e-10
1	TSLA	2026-02-20	110.0	417.070007	0.021918	9.394047	0.950962	0.950959	-0.000003	0.000175	0.000175	9.434257e-09	6.270220	6.270220	-2.967049e-10
2	TSLA	2026-02-20	120.0	417.070007	0.021918	3.609376	0.995339	0.995336	-0.000002	0.000061	0.000061	1.867849e-08	0.838722	0.838722	1.244740e-09
3	TSLA	2026-02-20	130.0	417.070007	0.021918	3.386720	0.995020	0.995017	-0.000003	0.000069	0.000069	2.289139e-08	0.889610	0.889610	1.594645e-09
4	TSLA	2026-02-20	140.0	417.070007	0.021918	3.250002	0.993980	0.993976	-0.000004	0.000085	0.000085	2.873577e-08	1.052985	1.052985	1.821572e-09
5	TSLA	2026-02-20	150.0	417.070007	0.021918	3.054690	0.993597	0.993592	-0.000004	0.000095	0.000096	3.495283e-08	1.112112	1.112112	1.687957e-09
6	TSLA	2026-02-20	155.0	417.070007	0.021918	2.960940	0.993413	0.993408	-0.000005	0.000101	0.000101	3.851312e-08	1.140339	1.140339	2.201295e-09
7	TSLA	2026-02-20	160.0	417.070007	0.021918	2.871097	0.993214	0.993209	-0.000005	0.000107	0.000107	4.244066e-08	1.170715	1.170715	2.210517e-09
8	TSLA	2026-02-20	165.0	417.070007	0.021918	2.835940	0.992250	0.992244	-0.000006	0.000122	0.000122	4.754362e-08	1.316233	1.316233	2.837598e-09
9	TSLA	2026-02-20	170.0	417.070007	0.021918	2.794925	0.991290	0.991283	-0.000006	0.000137	0.000137	5.294969e-08	1.458370	1.458370	2.692697e-09
10	TSLA	2026-02-20	175.0	417.070007	0.021918	2.617191	0.992599	0.992592	-0.000006	0.000127	0.000127	5.677028e-08	1.263888	1.263888	3.235334e-09
11	TSLA	2026-02-20	180.0	417.070007	0.021918	2.628910	0.990755	0.990747	-0.000008	0.000153	0.000153	6.412864e-08	1.536542	1.536542	2.618790e-09
12	TSLA	2026-02-20	185.0	417.070007	0.021918	2.585941	0.989743	0.989735	-0.000008	0.000171	0.000171	7.105261e-08	1.682487	1.682487	2.779016e-09
13	TSLA	2026-02-20	190.0	417.070007	0.021918	2.507816	0.989428	0.989419	-0.000009	0.000181	0.000181	7.819888e-08	1.727407	1.727407	3.127992e-09
14	TSLA	2026-02-20	195.0	417.070007	0.021918	2.429691	0.989150	0.989140	-0.000010	0.000191	0.000191	8.607820e-08	1.766940	1.766940	3.092089e-09
15	TSLA	2026-02-20	200.0	417.070007	0.021918	2.355473	0.988819	0.988808	-0.000011	0.000202	0.000202	9.482647e-08	1.813804	1.813804	3.358331e-09
16	TSLA	2026-02-20	210.0	417.070007	0.021918	2.175786	0.989094	0.989082	-0.000012	0.000214	0.000214	1.148914e-07	1.774918	1.774918	4.397929e-09
17	TSLA	2026-02-20	220.0	417.070007	0.021918	2.070317	0.987541	0.987526	-0.000015	0.000252	0.000253	1.406242e-07	1.992327	1.992327	4.362427e-09
18	TSLA	2026-02-20	225.0	417.070007	0.021918	1.974610	0.988089	0.988073	-0.000015	0.000255	0.000255	1.554089e-07	1.916161	1.916161	5.597368e-09
19	TSLA	2026-02-20	230.0	417.070007	0.021918	1.966797	0.985822	0.985804	-0.000018	0.000297	0.000297	1.720388e-07	2.227668	2.227668	4.554237e-09

- Both Greeks calculated using the Black–Scholes and finite-difference methods are very close, with very small differences, which means the finite-difference method gives correct estimations.
- There are also small differences in Delta, which means the option price moves closely with the underlying stock.
- Gamma and Vega are alos small in terms of differences but increases as strikes move closer toward the money.

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.

In [27]:
```python
#Removing duplicates
iv_map = (IV_values[["ticker", "exp", "strike", "type", "impliedVolatility"]].drop_duplicates(["ticker", "exp", "strike", "type"]))
#Map stock data
spot2 = dict(zip(DATA2["ticker"], DATA2["Close"]))
#Merging DATA1 IVs with DATA2 options
opt2 = (IV_values.copy().merge(iv_map, on=["ticker", "exp", "strike", "type"], how="left").assign(S0=lambda df: df["ticker"].map(spot2),
            r=r,T=lambda df: (pd.to_datetime(df["exp"]) - pd.Timestamp("2026-02-13")).dt.days / 365).loc[lambda df: df["T"] > 0]
        .copy())

#calculating option prices
opt2["BS_price_DATA2"] = opt2.apply(
    lambda row: (BlackScholesModel(S=row["S0"],
                                  K=row["strike"], T=row["T"], r=row["r"], sigma=row["IV_bisection"]).call_option_price()
```

```
                    if row["type"] == "call"
                    else BlackScholesModel(S=row["S0"], K=row["strike"],
                                           T=row["T"], r=row["r"], sigma=row["IV_bisection"]).put_option_price()),axis=1)
#removing NAs
opt2 = opt2.dropna(subset=["BS_price_DATA2"])
```

In [28]:

```
#dropping not needed columns
cols_to_drop = [
    'bid', 'ask', 'change', 'percentChange', 'volume', 'openInterest',
    'impliedVolatility_x', 'inTheMoney', 'contractSize', 'currency', 'ticker',
    'T', 'IV', 'mny', 'IV_newton', 'impliedVolatility_y', 'r'
]

BS_option_prices = opt2.drop(columns=cols_to_drop, errors="ignore").copy()
BS_option_prices
```

Out[28]:

	contractSymbol	lastTradeDate	strike	lastPrice	exp	type	Time_to_Maturity	mid	S0	IV_bisection	BS_price_DATA2
23	TSLA260220C00250000	2026-02-13 15:20:44+00:00	250.0	165.00	2026-02-20	call	0.013699	167.200	417.440002	1.335480	167.661320
25	TSLA260220C00260000	2026-02-12 19:23:39+00:00	260.0	157.85	2026-02-20	call	0.013699	157.800	417.440002	1.993320	159.200082
26	TSLA260220C00265000	2026-02-09 16:04:52+00:00	265.0	153.39	2026-02-20	call	0.013699	152.725	417.440002	1.882154	154.029003
27	TSLA260220C00270000	2026-02-13 18:11:32+00:00	270.0	150.24	2026-02-20	call	0.013699	147.400	417.440002	1.584215	148.266457
28	TSLA260220C00275000	2026-02-13 18:41:49+00:00	275.0	147.79	2026-02-20	call	0.013699	142.450	417.440002	1.566906	143.383945
...	...	...	...	...	...	...	...	...	...	...	...
2267	SPY260515P00775000	2026-02-11 20:58:05+00:00	775.0	80.87	2026-05-15	put	0.243836	93.420	681.750000	0.234048	93.108092
2268	SPY260515P00785000	2026-02-06 20:50:47+00:00	785.0	91.74	2026-05-15	put	0.243836	103.420	681.750000	0.251155	103.111401
2269	SPY260515P00790000	2026-02-06 20:50:47+00:00	790.0	96.76	2026-05-15	put	0.243836	108.420	681.750000	0.259538	108.113097
2270	SPY260515P00795000	2026-02-04 20:50:24+00:00	795.0	105.52	2026-05-15	put	0.243836	113.420	681.750000	0.267814	113.114861
2271	SPY260515P00800000	2026-02-04 20:50:24+00:00	800.0	110.54	2026-05-15	put	0.243836	118.420	681.750000	0.275989	118.116628

2013 rows × 11 columns

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 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:

• xt: BTC reserves at time t

• yt: USDC reserves at time t

• pool mid price is calculated as $Pt = yt/xt$ (USDC per BTC) • external market price St (USDC per BTC)

• fee rate $gamma$ (0; 1)

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

$$xt + 1yt + 1 = xtyt = k$$

(a) (10 pts) Derive the swap amounts

Case 1: $S > \dfrac{P_t}{1-\gamma}$ (BTC cheaper in the pool), Arbitrager swaps **USDC** $\to$ **BTC**.

Reserve updates based on the below:

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

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

Given Boundary conditions:

$\frac{y_{t+1}}{x_{t+1}} \cdot \frac{1}{1-\gamma} = S$ which we can rewrite as $\frac{y_{t+1}}{x_{t+1}} = S(1-\gamma)$

Solve for x_{t+1}:

$x_{t+1}y_{t+1} = k$, $y_{t+1} = x_{t+1}\, S(1-\gamma)$.

$$x_{t+1}\big(x_{t+1}S(1-\gamma)\big) = k \quad \Longrightarrow \quad x_{t+1}^2\, S(1-\gamma) = k.$$

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

Then

$$y_{t+1} = x_{t+1}S(1-\gamma) = S(1-\gamma)\sqrt{\frac{k}{S(1-\gamma)}} = \sqrt{kS(1-\gamma)}.$$

Swap sizes From $x_{t+1} = x - \Delta x$ will be $\Delta x = x - x_{t+1} = x - \sqrt{\frac{k}{S(1-\gamma)}}$.

From $y_{t+1} = y + (1-\gamma)\Delta y$ it will be $(1-\gamma)\Delta y = y_{t+1} - y$ which will be $\Delta y = \frac{y_{t+1}-y}{1-\gamma} = \frac{\sqrt{kS(1-\gamma)}-y}{1-\gamma}$.

Case 2: $S < P_t(1-\gamma)$ (BTC cheaper outside). Arbitrager swaps **BTC** $\to$ **USDC**

Reserve updates based on the below

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

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

Given Boundary conditions:

$\frac{y_{t+1}}{x_{t+1}}(1-\gamma) = S$ which we can rewrite as $\frac{y_{t+1}}{x_{t+1}} = \frac{S}{1-\gamma}$

Solve for x_{t+1}:

$$y_{t+1} = x_{t+1}\frac{S}{1-\gamma}$$

$$x_{t+1}\left(x_{t+1}\frac{S}{1-\rho}\right) = k$$

$$x_{t+1}^2 \frac{S}{1-\gamma} = k$$

Which will be

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

Now solve for y_{t+1}:

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

Swap sizes

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

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

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

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

$$\Delta y = y - y_{t+1} = y - \sqrt{\frac{kS}{1-\gamma}}$$

(b) (10 pts) Expected fee revenue

- I set L = 8 as all normal distribution is within +-8 STD.
- Step size N will be 20K to precisely capture the changes in the payoff.
- Given sigma = 0.2 and fee = 0.003

In [29]:
```
# As given data
sigma = 0.2 # vol
gamma   = 0.003 # Fee Rate
x  = 1000
y  = 1000
k  = x * y
dt = 1 / 365
Pt = y / x

# Trapezoid data
L = 8 #For the normal distribution
N = 20000 #Step size
h = (2 * L) / N

# Setting up the GBM
mu = -0.5 * sigma * sigma * dt
random_part = sigma * math.sqrt(dt)


upper_band = Pt / (1 - gamma)
lower_band = Pt * (1 - gamma)
```

```
total = 0
#Loop over grid points, from -L to +L
for i in range(N + 1):
    z = -L + i * h
    phi = (1 / math.sqrt(2 * math.pi)) * math.exp(-0.5 * z * z)
    s = math.exp(mu + random_part * z) #next price

    if s > upper_band:
        ynext = math.sqrt(k * s * (1 - gamma))
        deltay = (ynext - y) / (1 - gamma)
        R = gamma * deltay

    elif s < lower_band:
        xnext = math.sqrt(k * (1 - gamma) / s)
        deltax = (xnext - x) / (1 - gamma)
        R = gamma * deltax * s
    else:
        R = 0

    g = R * phi
    # rule weight - half weight at endpoints
    w = 0.5 if (i == 0 or i == N) else 1
    total += w * g

Expected_value = h * total
print("E[R] =", Expected_value)
```

```
E[R] = 0.008522036457782793
```

- The Expected fee revenue is 0.0085

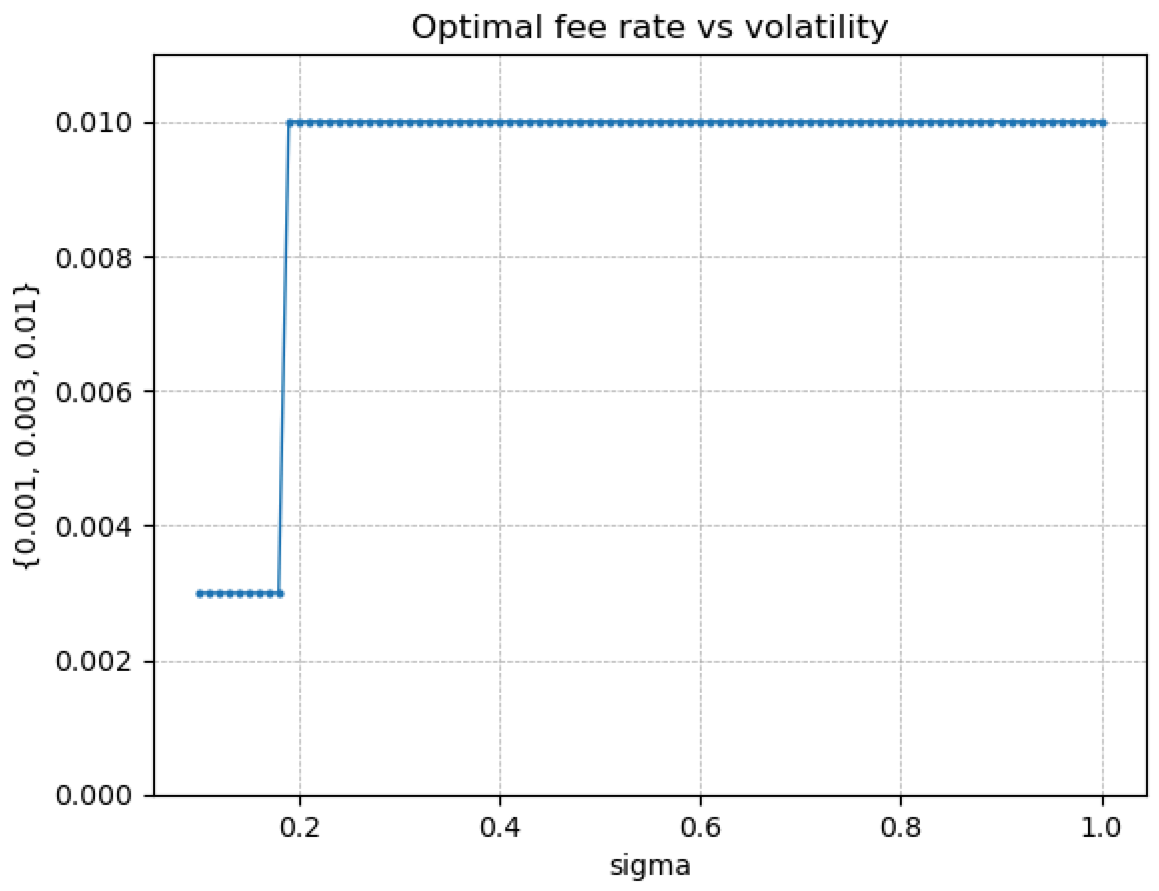
(c) (10 pts) Optimal Fee Rate under different volatilities

In [30]:

```
sigmas = [0.2, 0.6, 1] # given data
gammas = [0.001, 0.003, 0.01]# given data

table_rows = [] # empty table

for sigma in sigmas:
    # GBM parameters for this sigma
    mu = -0.5 * sigma * sigma * dt
    random_part = sigma * math.sqrt(dt)

    #E[R] for gammas
    ER_for_gammas = {}

    for gamma in gammas:
        # define boundaries
        upper_band = Pt / (1 - gamma)
        lower_band = Pt * (1 - gamma)

        total = 0
        for i in range(N + 1): #integration over the normal distribution
            z = -L + i * h

            phi = (1 / math.sqrt(2 * math.pi)) * math.exp(-0.5 * z * z)
            s = math.exp(mu + random_part * z) #next price

            if s > upper_band:
                ynext = math.sqrt(k * s * (1 - gamma))
                deltay = (ynext - y) / (1 - gamma)
                R = gamma * deltay

            elif s < lower_band:
                xnext = math.sqrt(k * (1 - gamma) / s)
                deltax = (xnext - x) / (1 - gamma)
                R = gamma * deltax * s

            else:
                R = 0

            g = R * phi
            w = 0.5 if (i == 0 or i == N) else 1
            total += w * g

        ER = h * total
        ER_for_gammas[gamma] = ER

    # choose best gamma for this sigma
    best_gamma = max(ER_for_gammas, key=ER_for_gammas.get)

    row = {"sigma": sigma}
    for gamma in gammas:
        row[f"E[R] (gamma={gamma})"] = ER_for_gammas[gamma]
    row["gamma*(sigma)"] = best_gamma
    table_rows.append(row)

df = pd.DataFrame(table_rows)
display(df)


# creating the GRID:

sigma_grid = [round(0.1 + 0.01*i, 2) for i in range(int(round((1 - 0.1)/0.01)) + 1)]
gamma_star = []

N_grid = 15000 # more integration to be accurate
h_grid = (2 * L) / N_grid

for sigma in sigma_grid:
    mu = -0.5 * sigma * sigma * dt
    random_part = sigma * math.sqrt(dt)

    ER_list = []

    for gamma in gammas:
        upper_band = Pt / (1 - gamma)
        lower_band = Pt * (1 - gamma)

        total = 0
        for i in range(N_grid + 1):
            z = -L + i * h_grid

            phi = (1 / math.sqrt(2 * math.pi)) * math.exp(-0.5 * z * z)
            s = math.exp(mu + random_part * z)

            if s > upper_band: #Computing payoff
                ynext = math.sqrt(k * s * (1 - gamma))
                deltay = (ynext - y) / (1 - gamma)
                R = gamma * deltay

            elif s < lower_band:
                xnext = math.sqrt(k * (1 - gamma) / s)
                deltax = (xnext - x) / (1 - gamma)
                R = gamma * deltax * s

            else:
                R = 0

            g = R * phi
            w = 0.5 if (i == 0 or i == N_grid) else 1
            total += w * g

        ER_list.append(h_grid * total)
    best_idx = max(range(len(gammas)), key=lambda j: ER_list[j])
    gamma_star.append(gammas[best_idx])
```

	sigma	E[R] (gamma=0.001)	E[R] (gamma=0.003)	E[R] (gamma=0.01)	gamma*(sigma)
0	0.2	0.003685	0.008522	0.009430	0.01
1	0.6	0.011923	0.032983	0.081082	0.01
2	1.0	0.020061	0.057384	0.160690	0.01

In [32]:
```
#creating plots
plt.figure()
plt.plot(sigma_grid, gamma_star, marker="o", markersize=2, linewidth=1)
plt.xlabel("sigma")
plt.ylabel("{0.001, 0.003, 0.01}")
plt.title("Optimal fee rate vs volatility")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.ylim(0.0, 0.011)
plt.show()
```



- As we can see as volatility/sigma goes up so as the fee goes up. Obviusly when volatility high then due to the way AMM wokrs there are more opportunities for arbitrage thus the fee is higher. Below 0.2 sigma when the vol is low then 0.003 seems to be the best fee.