Homework 1

FE621 Computational Finance

Joseph Gil

For questions that just need code without the output, I just put the code. For questions with an output, I have the code below the output.

Part 1:

1 & 2)

```
# PROBLEM 1 & 2
def get_market_data(tickers, d1, d2):
    results = {'DATA1': {}, 'DATA2': {}}
    for symbol in tickers:
        tkr = yf.Ticker(symbol)

        # Finds historical spot prices
        hist = tkr.history(start=d1, end="2026-02-15") # Ensure we capture up to today

        if d1 in hist.index:
            results['DATA1'][symbol] = {'spot': hist.loc[d1, 'Close']}
        if d2 in hist.index:
            results['DATA2'][symbol] = {'spot': hist.loc[d2, 'Close']}

        # Finds third Friday
        if symbol != "^VIX":
            monthly_expiries = []
            for exp in tkr.options:
                dt = pd.to_datetime(exp)
                if dt.weekday() == 4 and 15 <= dt.day <= 21:
                    monthly_expiries.append(exp)

            expiries = monthly_expiries[:3]

            chains = {}
            for exp in expiries:
                opt = tkr.option_chain(exp)
                chains[exp] = {'calls': opt.calls, 'puts': opt.puts}

            results['DATA1'][symbol]['options'] = chains
            results['DATA2'][symbol]['options'] = chains

    return results
```

```
# Define tickers and target dates
tickers = ['TSLA', 'SPY', '^VIX']
DATE1_STR = "2026-02-12"
DATE2_STR = "2026-02-13"

market_data = get_market_data(tickers, DATE1_STR, DATE2_STR)
DATA1 = market_data['DATA1']
DATA2 = market_data['DATA2']

print("Selected Monthly Expirations for SPY:", list(DATA1['SPY']['options'].keys()))
```

3)

The symbols we are using in this assignment are TSLA, SPY, and ^VIX. TSLA is the ticker symbol for Tesla Inc., the electric vehicle manufacturing company. SPY is the ticker symbol for the SPDR S&P 500 EFT. An ETF, or exchange-traded fund is a pooled investment security that trades on the stock exchange like a single stock. SPY tracks the S&P 500 Index, which is an index of the 500 largest companies in the United States. Lastly, ^VIX represents the CBOE Volatility Index. This index measures the expected 30-day volatility based on the S&P 500 option prices, and it can be used to discern investor anxiety. For option symbols, the ticker represents the underlying asset. The expiration date is a set of 6 digits formatted as YYMMDD to indicate the option's expiration. The option type symbol is a "C" for calls and a "P" for puts. Finally, the strike price is represented by a set of 8 digits that tells the price in thousands. So, 00100000 would represent a strike price of $100.00. Options generally expire monthly, on the third Friday of each month, but some highly liquid options can also expire weekly.

4)

Data recorded for 2026-02-12 (DATA1) and 2026-02-13 (DATA2).

Asset Prices at download (DATA1): TSLA: 417.07, SPY: 681.27, ^VIX: 20.82

FRED Short term interest rate = 3.64%

Prices and Maturities recorded for: 2026-02-12

TSLA Spot Price: 417.07

- Expiry: 2026-02-20 | Days to Maturity: 8 | T (Years): 0.0219
- Expiry: 2026-03-20 | Days to Maturity: 36 | T (Years): 0.0986
- Expiry: 2026-04-17 | Days to Maturity: 64 | T (Years): 0.1753

SPY Spot Price: 681.27

- Expiry: 2026-02-20 | Days to Maturity: 8 | T (Years): 0.0219
- Expiry: 2026-03-20 | Days to Maturity: 36 | T (Years): 0.0986
- Expiry: 2026-04-17 | Days to Maturity: 64 | T (Years): 0.1753

Prices and Maturities recorded for: 2026-02-13

TSLA Spot Price: 417.44

- Expiry: 2026-02-20 | Days to Maturity: 7 | T (Years): 0.0192
- Expiry: 2026-03-20 | Days to Maturity: 35 | T (Years): 0.0959
- Expiry: 2026-04-17 | Days to Maturity: 63 | T (Years): 0.1726

SPY Spot Price: 681.75

- Expiry: 2026-02-20 | Days to Maturity: 7 | T (Years): 0.0192
- Expiry: 2026-03-20 | Days to Maturity: 35 | T (Years): 0.0959
- Expiry: 2026-04-17 | Days to Maturity: 63 | T (Years): 0.1726

```python
# PROBLEM 4
print(" SOLUTION TO PROBLEM 4 ")
print(f"Data recorded for {DATE1_STR} (DATA1) and {DATE2_STR} (DATA2).")
print(f"Asset Prices at download (DATA1): TSLA: {DATA1['TSLA']['spot']:.2f}, SPY: {DATA1['SPY']['spot']:.2f}, ^VIX: {DATA1['^VIX']['spot']:.2f}")

# Interest rate form FRED
INT_Rate = 0.0364

for date_str, dataset in [("2026-02-12", DATA1), ("2026-02-13", DATA2)]:
    print(f"\nPrices and Maturities recorded for: {date_str}")
    for ticker in ['TSLA', 'SPY']:
        spot = dataset[ticker]['spot']
        print(f"  {ticker} Spot Price: {spot:.2f}")

        # Calculate T for each expiration date in the dataset and print it in a chart
        for exp in dataset[ticker]['options'].keys():
            dt_expiry = pd.to_datetime(exp)
            dt_current = pd.to_datetime(date_str)
            days_diff = (dt_expiry - dt_current).days
            T_years = max(days_diff / 365.0, 0.0001)

            print(f"    - Expiry: {exp} | Days to Maturity: {days_diff} | T (Years): {T_years:.4f}")
```

Part 2:

5)

```python
# PROBLEM 5:
def bs_price(S0, K, T, r, sigma, opt_type='call'):        # Function that takes input values and runs the Black-Scholes Formula
    if sigma <= 0 or T <= 0: return 0
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if opt_type == 'call':
        return S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    else:
        return K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)
```

6)

ATM Implied Volatilities (closest to 1.0 moneyness):

TSLA (Call): 36.7332% at Strike 417.5

SPY (Call): 16.6631% at Strike 681.0

7)

Bisection Method Avg Time: 0.00615442 seconds

Newton Method Avg Time: 0.00165764 seconds

Newton is 3.7x faster than Bisection in this dataset.

#Code is combined between 6 and 7

```python
# PROBLEM 6 & 7:
print("\n SOLUTION TO PROBLEM 6 & 7: ")

all_iv_data = []
bisection_times = []
newton_times = []

for ticker in ['TSLA', 'SPY']:
    S0 = DATA1[ticker]['spot']

    options_dict = DATA1[ticker].get('options', {})

    for exp, chain in options_dict.items():               # Calculates Time to Maturity in years
        days_to_expiry = (pd.to_datetime(exp) - pd.to_datetime(DATE1_STR)).days
        T = max(days_to_expiry / 365.0, 0.001)


        for opt_type_key in ['calls', 'puts']:       # Iterate through both calls and puts
            options_df = chain[opt_type_key]

            # Sets up variables and labels depending on if it is a call or a put
            func_label = 'call' if opt_type_key == 'calls' else 'put'
            display_label = 'Call' if opt_type_key == 'calls' else 'Put'

            # Filters in the money options using 0.9 and 1.1
            filtered = options_df[(options_df['volume'] > 0) &
                                  (S0 / options_df['strike'] >= 0.9) &
                                  (S0 / options_df['strike'] <= 1.1)]

            for _, opt in filtered.iterrows():
                K = opt['strike']
                market_price = (opt['bid'] + opt['ask']) / 2


                # Solves Problem 6 using the bisection method
                t0 = time.time()
                iv_b = bisection_iv(market_price, S0, K, T, INT_RATE, func_label)
                bisection_times.append(time.time() - t0)


                # Solves Problem 7 using the Newton method
                t1 = time.time()
                iv_n = newton_iv(market_price, S0, K, T, INT_RATE, func_label)
                newton_times.append(time.time() - t1)

                # Stores the data for Problem 8
                all_iv_data.append({
                    'Ticker': ticker,
                    'Maturity': exp,
                    'Type': display_label,
                    'Strike': K,
                    'Price': market_price,
                    'IV': iv_b,
                    'Moneyness': S0 / K
                })

# Create the DataFrame for Problem 8
iv_df = pd.DataFrame(all_iv_data)

# Find and print Problem 6 results
atm_results = []
print("ATM Implied Volatilities (closest to 1.0 moneyness):")
for ticker in ['TSLA', 'SPY']:
    subset = iv_df[iv_df['Ticker'] == ticker]
    if not subset.empty:
        atm_idx = (subset['Moneyness'] - 1).abs().idxmin()
        atm_opt = subset.loc[atm_idx]
        atm_results.append(atm_opt)
        print(f"{ticker} ({atm_opt['Type']}): {atm_opt['IV']:.4%} at Strike {atm_opt['Strike']}")

# Print Problem 7 results
avg_bis = np.mean(bisection_times) if bisection_times else 0
avg_new = np.mean(newton_times) if newton_times else 0
print(f"Bisection Method Avg Time: {avg_bis:.8f} seconds")
print(f"Newton Method Avg Time:    {avg_new:.8f} seconds")
if avg_new > 0:
    print(f"Newton is {avg_bis/avg_new:.1f}x faster than Bisection in this dataset.\n")
```

8)

Ticker	Maturity	Option Type	Avg Implied Volatility
SPY	2026-02-20	Call	19.09%
SPY	2026-02-20	Put	18.68%
SPY	2026-03-20	Call	16.65%

SPY 2026-03-20	Put	19.22%
SPY 2026-04-17	Call	14.89%
SPY 2026-04-17	Put	17.89%
TSLA 2026-02-20	Call	38.03%
TSLA 2026-02-20	Put	34.79%
TSLA 2026-03-20	Call	43.27%
TSLA 2026-03-20	Put	42.48%
TSLA 2026-04-17	Call	44.57%
TSLA 2026-04-17	Put	43.92%

Comparison with ^VIX: The ^VIX was 20.82. The SPY ATM IV is approximately 16.66%.

The implied volatilities of TSLA and SPY are consistent with the volatility of the stock. All of the implied volatilities of the SPY options are much less than those of the TSLA options. This makes sense, since SPY options are generally risk-averse. When maturity increases, TSLA has an increased implied volatility while SPY has a decreased implied volatility. Also, the TSLA volatility decreases with in the moneyness while the SPY volatility increases. The calculated ^VIX is pretty close to the SYP ATM IV.

```
# PROBLEM 8:
print("\n SOLUTION TO PROBLEM 8: ")

# Creates and prints the summary table
summary_table = iv_df.groupby(['Ticker', 'Maturity', 'Type'])['IV'].mean().reset_index()
summary_table.columns = ['Ticker', 'Maturity', 'Option Type', 'Avg Implied Volatility']
summary_table['Avg Implied Volatility'] = summary_table['Avg Implied Volatility'].map('{:.2%}'.format)

print(summary_table.to_string(index=False))
```

9)

Put-Call Parity Check for SPY (Exp: 2026-02-20):

Strike	Call_Mkt	Call_Calc	Put_Mkt	Put_Calc	Diff
649.0000	33.9200	33.4800	0.6900	1.1400	0.4400

650.0000 32.9800 32.5300 0.7400 1.1900 0.4500

651.0000 32.0100 31.5800 0.8000 1.2200 0.4200

652.0000 31.0800 30.6400 0.8500 1.2900 0.4400

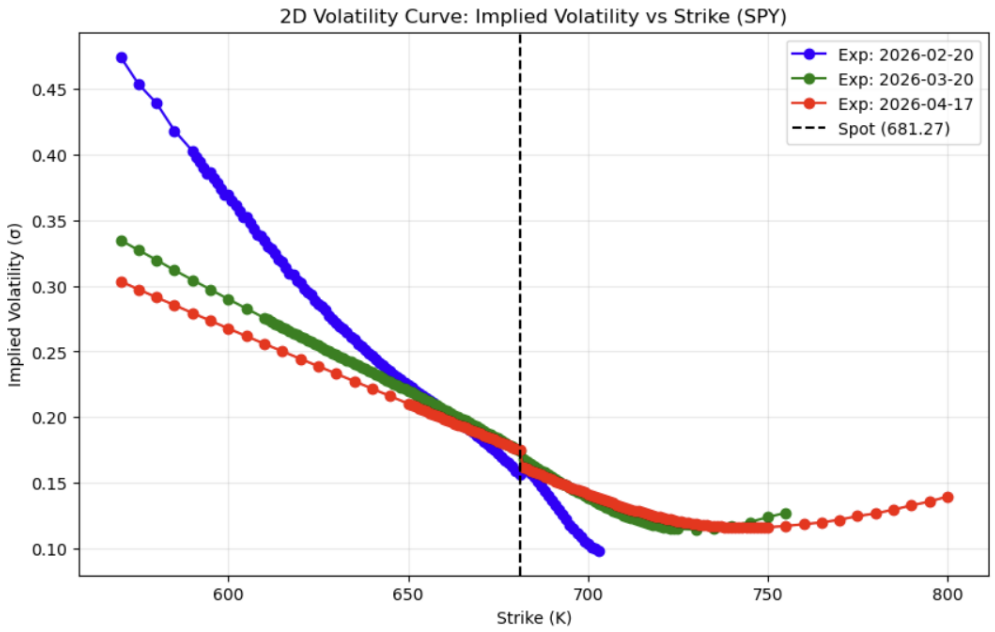
653.0000 30.1500 29.7000 0.9100 1.3500 0.4400

Put-Call Parity Check for TSLA (Exp: 2026-02-20):

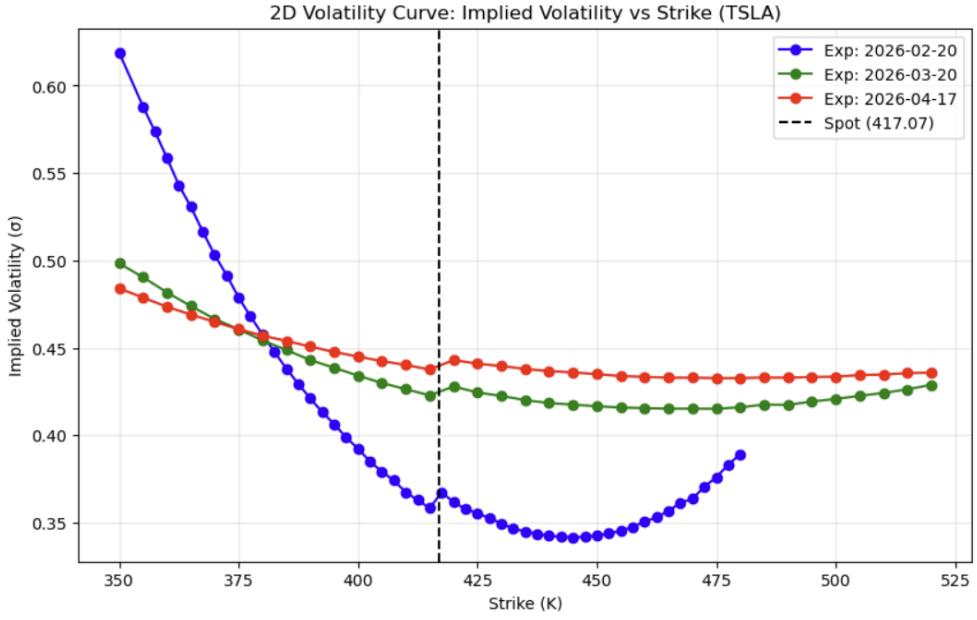
Strike	Call_Mkt	Call_Calc	Put_Mkt	Put_Calc	Diff
397.5000	23.0700	22.6400	2.7500	3.1900	0.4400
400.0000	20.9300	20.5900	3.2000	3.5400	0.3400
402.5000	18.9500	18.5900	3.7000	4.0600	0.3600
405.0000	17.0500	16.6900	4.3000	4.6600	0.3600
407.5000	15.2500	14.8900	5.0000	5.3600	0.3600

```
# PROBLEM 9:
print("\n SOLUTION TO PROBLEM 9: ")

for ticker in ['SPY', 'TSLA']:
    S0 = DATA1[ticker]['spot']
    closest_exp = list(DATA1[ticker]['options'].keys())[0]

    T = max((pd.to_datetime(closest_exp) - pd.to_datetime(DATE1_STR)).days / 365.0, 0.001)

    calls = DATA1[ticker]['options'][closest_exp]['calls']
    puts = DATA1[ticker]['options'][closest_exp]['puts']

    # Merges calls and puts on Strike to compare them
    merged = pd.merge(calls, puts, on='strike', suffixes=('_call', '_put'))
    merged = merged[(merged['volume_call'] > 0) & (merged['volume_put'] > 0)]
    merged = merged[(S0/merged['strike'] >= 0.95) & (S0/merged['strike'] <= 1.05)]

    parity_results = []
    for _, row in merged.iterrows():
        K = row['strike']
        c_mkt = (row['bid_call'] + row['ask_call']) / 2
        p_mkt = (row['bid_put'] + row['ask_put']) / 2

        # Calculates C and P in C - P = S - K*e^(-rT)
        p_calc = c_mkt - S0 + K * np.exp(-INT_RATE * T)
        c_calc = p_mkt + S0 - K * np.exp(-INT_RATE * T)

        parity_results.append({
            'Strike': K, 'Call_Mkt': round(c_mkt, 2), 'Call_Calc': round(c_calc, 2),
            'Put_Mkt': round(p_mkt, 2), 'Put_Calc': round(p_calc, 2),
            'Diff': round(abs(p_calc - p_mkt), 2)
        })

    # Prints the Put-Call Parity charts for SPY and TSLA
    print(f"\nPut-Call Parity Check for {ticker} (Exp: {closest_exp}):")
    print(pd.DataFrame(parity_results).head(5).to_string(index=False))
```

10)



3D Volatility Surface (SPY)

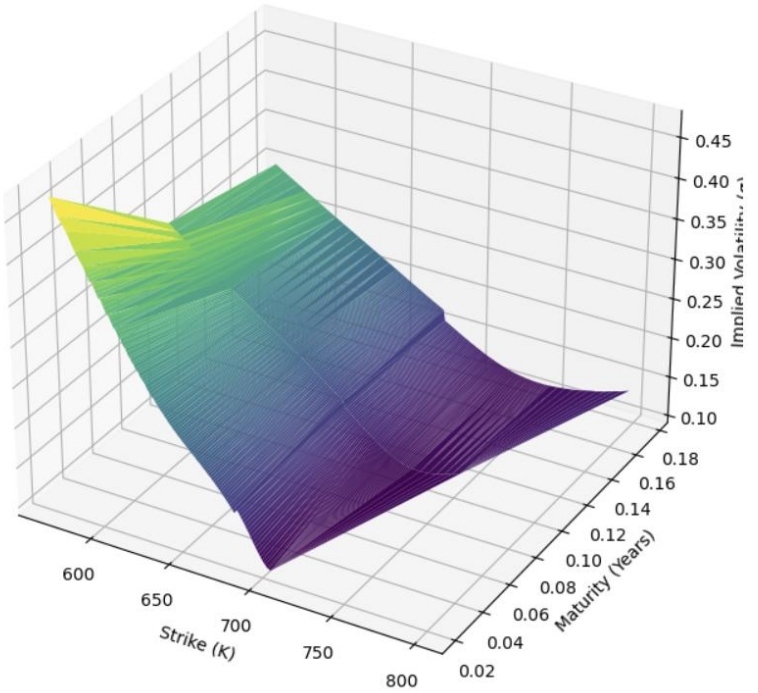


3D Volatility Surface (TSLA)

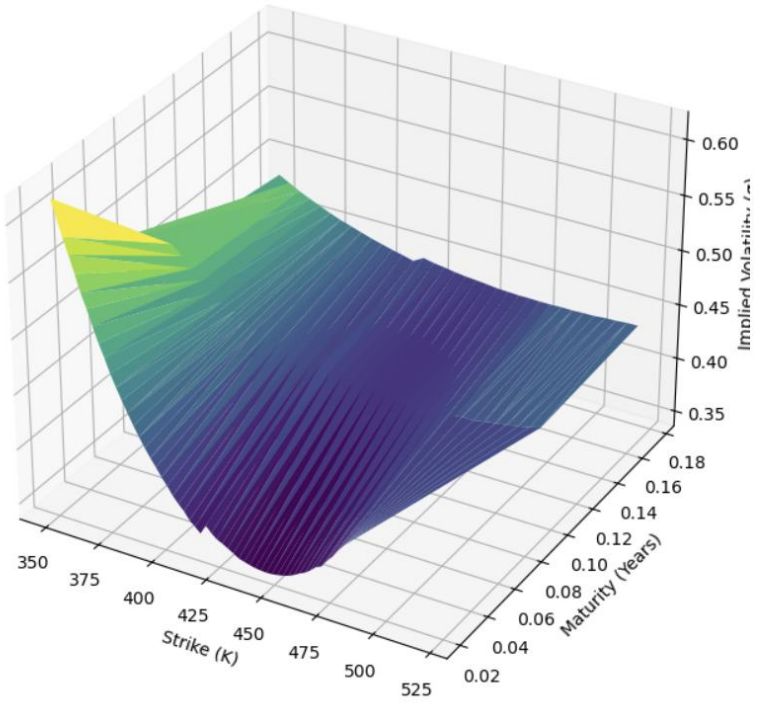
```python
# PROBLEM 10:

print("\nSOLUTION TO PROBLEM 10: ")

all_plot_data = {'SPY': [], 'TSLA': []}

for ticker in ['SPY', 'TSLA']:
    S0 = DATA1[ticker]['spot']
    expirations = list(DATA1[ticker]['options'].keys())

    for exp in expirations:
        T = max((pd.to_datetime(exp) - pd.to_datetime(DATE1_STR)).days / 365.0, 0.001)

        calls = DATA1[ticker]['options'][exp]['calls']
        puts = DATA1[ticker]['options'][exp]['puts']

        # OTM Puts
        for _, row in puts[puts['strike'] < S0].iterrows():
            K = row['strike']
            price = (row['bid'] + row['ask']) / 2
            if price > 0.05 and S0/K <= 1.2:
                iv = newton_iv(price, S0, K, T, INT_RATE, 'put')
                if not np.isnan(iv) and 0 < iv < 2.0:
                    all_plot_data[ticker].append({'Exp': exp, 'T': T, 'Strike': K, 'IV': iv})

        # OTM Calls
        for _, row in calls[calls['strike'] >= S0].iterrows():
            K = row['strike']
            price = (row['bid'] + row['ask']) / 2
            if price > 0.05 and S0/K >= 0.8:
                iv = newton_iv(price, S0, K, T, INT_RATE, 'call')
                if not np.isnan(iv) and 0 < iv < 2.0:
                    all_plot_data[ticker].append({'Exp': exp, 'T': T, 'Strike': K, 'IV': iv})

    plot_df = pd.DataFrame(all_plot_data[ticker])
```

```python
    # Plotting the 2D Graph
    plt.figure(figsize=(10, 6))
    colors = ['blue', 'green', 'red']
    for i, exp in enumerate(expirations):
        subset = plot_df[plot_df['Exp'] == exp].sort_values('Strike')
        plt.plot(subset['Strike'], subset['IV'], marker='o', linestyle='-', color=colors[i], label=f'Exp: {exp}')

    plt.axvline(x=S0, color='black', linestyle='--', label=f'Spot ({S0:.2f})')
    plt.title(f'2D Volatility Curve: Implied Volatility vs Strike ({ticker})')
    plt.xlabel('Strike (K)')
    plt.ylabel('Implied Volatility (σ)')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

    # Plotting the 3D Graph
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_trisurf(plot_df['Strike'], plot_df['T'], plot_df['IV'], cmap='viridis', edgecolor='none')
    ax.set_title(f'3D Volatility Surface ({ticker})')
    ax.set_xlabel('Strike (K)')
    ax.set_ylabel('Maturity (Years)')
    ax.set_zlabel('Implied Volatility (σ)')
    plt.show()
```

Extra Credit:



3D Volatility Surface (TSLA)



11)

Ticker	Strike	Delta(BS)	Delta(Ap)	Gamma(BS)	Gamma(Ap)	Vega(BS)	Vega(Ap)
SPY	681.0000	0.5242	0.5242	0.0237	0.0237	40.1630	40.1630
TSLA	417.5000	0.5091	0.5091	0.0176	0.0176	24.6266	24.6266

The Greeks from the two methods are practically identical to each other.a

```
# PROBLEM 11:

print("\nSOLUTION TO PROBLEM 11: ")

# Approximate greeks calculations
def calc_greeks_approx(S0, K, T, r, sigma, dS=0.01, dVol=0.0001):
    p_up_S = bs_price(S0 + dS, K, T, r, sigma, 'call')
    p_dn_S = bs_price(S0 - dS, K, T, r, sigma, 'call')
    delta_ap = (p_up_S - p_dn_S) / (2 * dS)

    p_base = bs_price(S0, K, T, r, sigma, 'call')
    gamma_ap = (p_up_S - 2 * p_base + p_dn_S) / (dS**2)

    p_up_V = bs_price(S0, K, T, r, sigma + dVol, 'call')
    p_dn_V = bs_price(S0, K, T, r, sigma - dVol, 'call')
    vega_ap = (p_up_V - p_dn_V) / (2 * dVol)

    return delta_ap, gamma_ap, vega_ap

greek_results = []
for ticker in ['SPY', 'TSLA']:
    S0 = DATA1[ticker]['spot']
    subset = iv_df[(iv_df['Ticker'] == ticker) & (iv_df['Type'] == 'Call')]
    atm_opt = subset.loc[(subset['Moneyness'] - 1).abs().idxmin()]

    K, T, iv = atm_opt['Strike'], atm_opt['Maturity'], atm_opt['IV']
    T_years = max((pd.to_datetime(T) - pd.to_datetime(DATE1_STR)).days / 365.0, 0.001)

    # Black-Scholes method
    d1 = (np.log(S0 / K) + (INT_RATE + 0.5 * iv**2) * T_years) / (iv * np.sqrt(T_years))
    d_an = norm.cdf(d1)
    g_an = norm.pdf(d1) / (S0 * iv * np.sqrt(T_years))
    v_an = S0 * np.sqrt(T_years) * norm.pdf(d1)
```

```
    # Approximation method
    d_ap, g_ap, v_ap = calc_greeks_approx(S0, K, T_years, INT_RATE, iv)

    greek_results.append({
        'Ticker': ticker, 'Strike': K,
        'Delta (BS)': d_an, 'Delta (Ap)': d_ap,
        'Gamma (BS)': g_an, 'Gamma (Ap)': g_ap,
        'Vega (BS)': v_an, 'Vega (Ap)': v_ap
    })

greeks_df = pd.DataFrame(greek_results)
pd.set_option('display.float_format', '{:.4f}'.format)
print(greeks_df.to_string(index=False))
```

12)

Ticker	Strike	Day2_Spot	IV_From_Day1	Calc_Day2_Price
SPY	681.0000	681.7500	16.66%	6.9000
TSLA	417.5000	417.4400	36.73%	8.5800

```
# PROBLEM 12:

print("\nSOLUTION TO PROBLEM 12: ")

day2_pricing = []
for ticker in ['SPY', 'TSLA']:
    S0_Day2 = DATA2[ticker]['spot']

    subset = iv_df[(iv_df['Ticker'] == ticker) & (iv_df['Type'] == 'Call')]
    atm_opt = subset.loc[(subset['Moneyness'] - 1).abs().idxmin()]

    K, exp, iv_day1 = atm_opt['Strike'], atm_opt['Maturity'], atm_opt['IV']
    T_years = max((pd.to_datetime(exp) - pd.to_datetime(DATE1_STR)).days / 365.0, 0.001)

    T_Day2 = T_years - (1.0 / 365.0)

    if T_Day2 > 0:
        bs_price_day2 = bs_price(S0_Day2, K, T_Day2, INT_RATE, iv_day1, 'call')

        day2_pricing.append({
            'Ticker': ticker,
            'Strike': K,
            'Day2_Spot': S0_Day2,
            'IV_From_Day1': f"{iv_day1:.2%}",
            'Calc_Day2_Price': round(bs_price_day2, 2)
        })

print(pd.DataFrame(day2_pricing).to_string(index=False))
```

Part 3:

a)

Case 1: $S_{t+1} > P_t \, 1/(1-\gamma)$

$P_t = (y_{t+1}/ x_{t+1})$

So, $S_{t+1} = (y_{t+1}/ x_{t+1}) * 1/(1-\gamma)$

Since $y_{t+1} * x_{t+1} = k$, $x_{t+1} = k/ y_{t+1}$

As a result, $y_{t+1}/(k/y_{t+1}) = S_{t+1}(1-\gamma) \rightarrow y_{t+1}^2 = k\, S_{t+1}(1-\gamma) \rightarrow y_t$

Since $y_{t+1} = y_t + (1-\gamma)\Delta y$, $\Delta y = (\text{Sqrt}(k\, S_{t+1}(1-\gamma)) - y_t)/(1-\gamma)$

Case 2: $S_{t+1} < P_t \, 1/(1-\gamma)$

$P_t = (y_{t+1}/ x_{t+1})$

So, $S_{t+1} = (y_{t+1}/ x_{t+1}) * 1/(1-\gamma)$

Since $y_{t+1} * x_{t+1} = k$, $y_{t+1} = k/ x_{t+1}$

As a result, $((k/x_{t+1})(1-\gamma))/(x_{t+1}) = S_{t+1} \rightarrow x_{t+1}^2 = (k(1-\gamma))/ S_{t+1} \rightarrow x_{t+1}$

Since $x_{t+1} = x_t + (1-\gamma)\Delta x$, $\Delta x = (\text{Sqrt}(k(1-\gamma)/S_{t+1}) - x_t)/(1-\gamma)$

b)

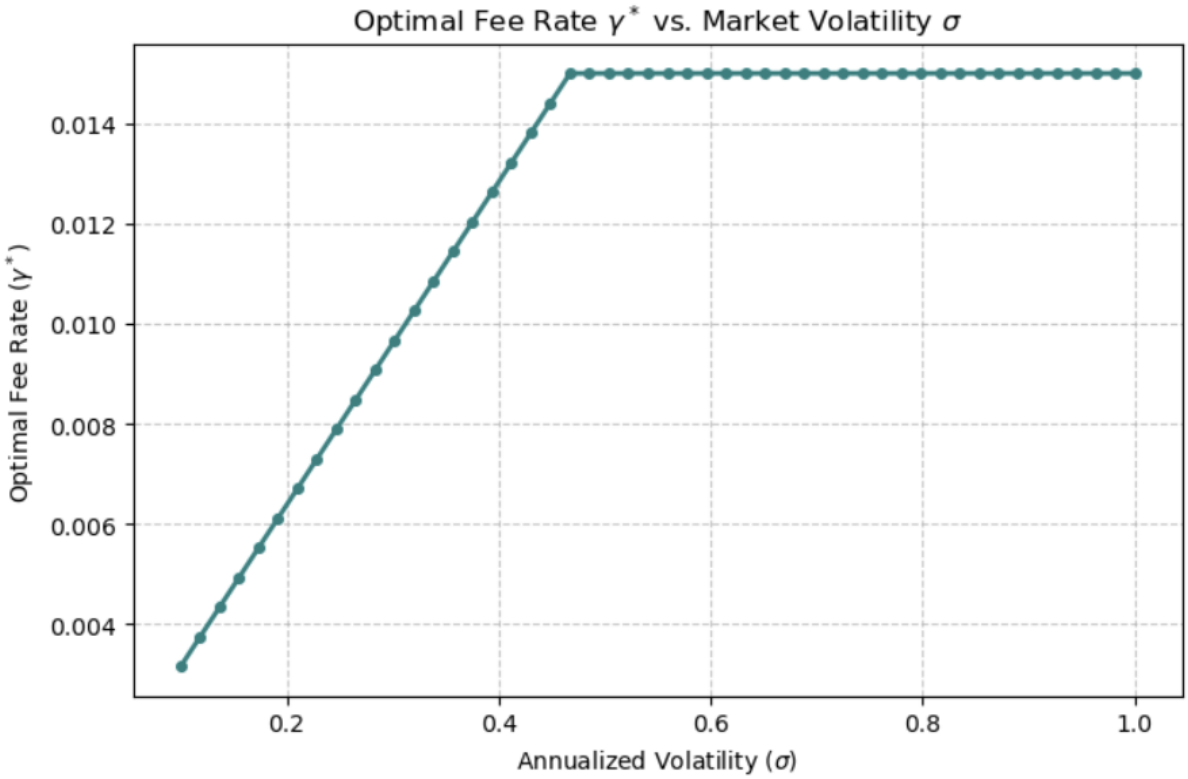
Expected One-Step Fee Revenue when sigma=0.2 and gamma=0.003: $0.00852204

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Constants
x_t = 1000.0
y_t = 1000.0
k = x_t * y_t
P_t = y_t / x_t
S_t = 1.0
dt = 1.0 / 365.0

# Function Definitions
def delta_y(S_next, gamma):
    return (np.sqrt(k * S_next * (1 - gamma)) - y_t) / (1 - gamma)

def delta_x(S_next, gamma):
    return (np.sqrt(k * (1 - gamma) / S_next) - x_t) / (1 - gamma)

def lognormal_pdf(s, sigma):
    mu = -0.5 * sigma**2 * dt
    v = sigma * np.sqrt(dt)
    return (1.0 / (s * v * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((np.log(s) - mu) / v)**2)

def expected_fee_revenue(gamma, sigma, n_points=10000):
    upper_limit = S_t * np.exp(6 * sigma * np.sqrt(dt))
    lower_limit = S_t * np.exp(-6 * sigma * np.sqrt(dt))

    threshold_up = P_t / (1 - gamma)
    threshold_down = P_t * (1 - gamma)

    # Case 1 Integration:
    if threshold_up < upper_limit:
        s_1 = np.linspace(threshold_up, upper_limit, n_points)
        dy = delta_y(s_1, gamma)
        integrand_1 = gamma * dy * lognormal_pdf(s_1, sigma)
        I1 = np.trapz(integrand_1, s_1)
    else: I1 = 0.0
```

```python
    # Case 2 Integration:
    if threshold_down > lower_limit:
        s_2 = np.linspace(lower_limit, threshold_down, n_points)
        dx = delta_x(s_2, gamma)
        integrand_2 = gamma * dx * s_2 * lognormal_pdf(s_2, sigma)
        I2 = np.trapz(integrand_2, s_2)
    else: I2 = 0.0

    return I1 + I2


# SOLUTION TO 3(b):
sigma_b = 0.2
gamma_b = 0.003
revenue_b = expected_fee_revenue(gamma_b, sigma_b)

print(f"PART 3(b)")
print(f"Expected One-Step Fee Revenue when sigma=0.2 and gamma=0.003: ${revenue_b:.8f}")
```

c)

Sigma	Optimal Gamma	Max E[R]
0.2000	0.0100	0.0094
0.6000	0.0100	0.0811
1.0000	0.0100	0.1607




```python
# SOLUTION TO 3(c):
print("\nPART 3(c)")
sigmas_test = [0.2, 0.6, 1.0]
gammas_test = [0.001, 0.003, 0.01]

opt_results = []
for s in sigmas_test:
    best_g = 0
    max_r = -1
    for g in gammas_test:
        r = expected_fee_revenue(g, s)
        if r > max_r:
            max_r = r
            best_g = g
    opt_results.append({'Sigma': s, 'Optimal Gamma': best_g, 'Max E[R]': max_r})

print(pd.DataFrame(opt_results).to_string(index=False))

# Grid for Plot
sigma_grid = np.linspace(0.1, 1.0, 50)
gamma_choices = np.linspace(0.0005, 0.015, 50)
optimal_gamma_curve = []

for s in sigma_grid:
    revs = [expected_fee_revenue(g, s) for g in gamma_choices]
    optimal_gamma_curve.append(gamma_choices[np.argmax(revs)])

# Plot for 3(c)
plt.figure(figsize=(8, 5))
plt.plot(sigma_grid, optimal_gamma_curve, color='teal', linewidth=2, marker='o', markersize=4)
plt.title('Optimal Fee Rate $\gamma^*$ vs. Market Volatility $\sigma$')
plt.xlabel('Annualized Volatility ($\sigma$)')
plt.ylabel('Optimal Fee Rate ($\gamma^*$)')
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```