FE 621: Computational Methods

Assignment 1

Bryan Takeshita

February 15, 2026

Part 1: Data Gathering Component

1.1: For this assignment, I used Yahoo Finance to collect my data.

```
import yfinance as yf
import pandas as pd
import datetime as dt
```

1.2: Next, we were given the tickers for the data. I created two dates to reflect the two data sets we needed to create as well as created the code to download the options.

```
day1 = "2026-02-12"
day2 = "2026-02-13"

def download_intraday(symbol, date):
    ticker = yf.Ticker(symbol)
    data = ticker.history(
        start=date,
        end=(pd.to_datetime(date) + pd.Timedelta(days=1)).strftime('%Y-%m-%d'),
        interval="5m"
    )
    return data

def get_spot_price(symbol):
    ticker = yf.Ticker(symbol)
    data = ticker.history(period="1d", interval="1m")
    return data["Close"].iloc[-1]
```

```
def download_option_chain(symbol, expirations):
    ticker = yf.Ticker(symbol)
    option_data = {}

    for exp in expirations:
        opt = ticker.option_chain(exp)
        option_data[exp] = {
            "calls": opt.calls,
            "puts": opt.puts
        }

    return option_data
```

The reason for so many maturities is due to the ability to exercise immediately. Multiple different maturities create a safeguard for liquidity risk as a trader could utilize the different maturities in their favor. Also, Traders have different goals when using these options. Some are long-term, some are short-term, some are risk-averse, while some are risk-neutral. The different maturities allow the traders to create a portfolio that suits their needs.

1.3: The SPY is an ETF that is created to mirror the S&P 500. The main purpose of the SPY is to replicate the earnings of the S&P 500 while also allowing for high liquidity. The SPY allows a trader to earn these earnings without having to own all 500 stocks in the S&P 500. The VIX is a real-time market index. The VIX measures the 30-day expectation of future volatility in the S&P 500 Index. The VIX is also known as the fear gauge. TSLA is a stock that allows a trader to invest in the company, Tesla. TSLA is one of the most traded actively traded stocks in the market, making it a very valuable asset due to its liquidity.

1.4: When I downloaded the data, the current price for the listed tickers were:

```
TSLA: 417.5050048828125
SPY : 680.77001953125
VIX : 20.34000015258789
```

Using the website attached to the assignment, I used the Federal Funds Effective for the date of the data. That value was 3.64%. I use this value in my calculations.

To calculate the Time to maturity, I took the expiration of my data found the difference between the day of maturity and the day of the download. Then I converted it into years.

```python
def compute_ttm(data_dict):

    # download
    download_time = data_dict["intraday"].index[-1]

    # timezone
    if download_time.tzinfo is not None:
        download_time = download_time.tz_localize(None)

    ttm_results = {}

    for exp in data_dict["expirations"]:

        expiration_date = pd.to_datetime(exp)

        # Set expiration to 4:00pm
        expiration_datetime = expiration_date.replace(hour=16, minute=0)

        # Compute TTM in years
        T = (expiration_datetime - download_time).total_seconds() / (365 * 24 * 60 * 60)

        ttm_results[exp] = T

    return ttm_results
```

```
TSLA:
Expiration 2026-02-20 -> TTM = 0.021927 years
Expiration 2026-03-20 -> TTM = 0.098640 years
Expiration 2026-04-17 -> TTM = 0.175352 years

SPY:
Expiration 2026-02-20 -> TTM = 0.021927 years
Expiration 2026-03-20 -> TTM = 0.098640 years
Expiration 2026-04-17 -> TTM = 0.175352 years
```

```
TSLA:
Expiration 2026-02-20 -> TTM = 0.019245 years
Expiration 2026-03-20 -> TTM = 0.095957 years
Expiration 2026-04-17 -> TTM = 0.172669 years

SPY:
Expiration 2026-02-20 -> TTM = 0.019245 years
Expiration 2026-03-20 -> TTM = 0.095957 years
Expiration 2026-04-17 -> TTM = 0.172669 years
```

Part 2: Analysis of Data

2.5: Using the Black-Scholes Formula, I create code with took in the normal inputs as well as the type of option to calculate the price.

```python
#r = 3.64% based on link

def black_scholes_price(S0, K, T, r, sigma, option_type):
    """
    Inputs:
        S0 : stock price at download
        K  : strike price
        T  : time to maturity (in years)
        r  : annual risk-free rate (decimal)
        sigma : annual volatility (decimal)
        option_type : 'call' or 'put'
    """
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    # Call or Put
    if option_type.lower() == "call":
        price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    else option_type.lower() == "put":
        price = K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)
    return price
```

2.6: Implementing the Bisection method, I got the results of:

```
TSLA ATM IV : 0.411966
TSLA Avg IV : 1.262101

SPY ATM IV  : 0.187219
SPY Avg IV  : 0.60496
```

Function for it:

```python
def compute_iv_for_asset(asset_name):

    exp = DATA1[asset_name]["expirations"][0]
    chain = DATA1[asset_name]["options"][exp]["calls"].copy()
    S0 = DATA1[asset_name]["spot_at_download"]
    T = compute_ttm(DATA1[asset_name])[exp]
    r = 0.0364

    iv_list = []
    for _, row in chain.iterrows():

        mid_price = (row["bid"] + row["ask"]) / 2
        K = row["strike"]
        iv = bisection_iv(S0, K, T, r, mid_price, "call")
        iv_list.append((K, iv))

    iv_df = pd.DataFrame(iv_list, columns=["strike","IV"])
    # Closest Strike to Spot
    atm_strike = iv_df.iloc[(iv_df["strike"] - S0).abs().argsort()[:1]]
    atm_iv = atm_strike["IV"].values[0]
    # Average
    avg_iv = iv_df["IV"].mean()
    return atm_iv, avg_iv, iv_df

atm_iv_tsla, avg_iv_tsla, iv_df_tsla = compute_iv_for_asset("TSLA")
atm_iv_spy, avg_iv_spy, iv_df_spy = compute_iv_for_asset("SPY")
```

2.7: For this part, I used the Newton Method. When I calculate the time to find the roots, I get the values:

```
TSLA Timing (seconds):
Avg Bisection: 0.003968208486383611
Avg Newton   : 0.0004924658573035038

SPY Timing (seconds):
Avg Bisection: 0.0024998622636000314
Avg Newton   : 0.0007433108985424042
```

As you can see, the Newton Method is faster than the bisection method. This is since the bisection method is linear while the Newton method is quadratic.

2.8:

```
  Asset   Maturity OptionType    ATM_IV  Average_IV
0  TSLA  2026-02-20      calls  0.411966    0.386393
1  TSLA  2026-02-20       puts  0.309017    0.195661
2  TSLA  2026-03-20      calls  0.446517    0.422305
3  TSLA  2026-03-20       puts  0.395519    0.329615
4  TSLA  2026-04-17      calls  0.456887    0.481518
5  TSLA  2026-04-17       puts  0.417317    0.428015
0   SPY  2026-02-20      calls  0.187219    0.234121
1   SPY  2026-02-20       puts  0.131824    0.168724
2   SPY  2026-03-20      calls  0.179502    0.214981
3   SPY  2026-03-20       puts  0.161799    0.209570
4   SPY  2026-04-17      calls  0.170335    0.183953
5   SPY  2026-04-17       puts  0.165400    0.209965
Current VIX: 20.34000015258789
```

The Implied Volatilities between Tesla and SPY are very constant with the overall volatility of the stock. The implied Volatilities of SPY are less than those of TSLA. This is what is expected since SPY is a very good risk-averse option. Also, we see the volatility change in and out of the money. For TSLA we see the volatility decrease in comparison to in the money. While SPY volatility increases in comparison to in the money. Also, TSLA sees an increase in volatility as maturity increases while SPY sees a decrease in volatility as the maturity increases. The VIX price calculated is very close to the given VIX price on the market.

2.9: The code created calculates using the mid or (Bid + Ask)/2.

```
# Put-Call Parity
synthetic_call = put_mid + S0 - K * discount
synthetic_put  = call_mid - S0 + K * discount
```

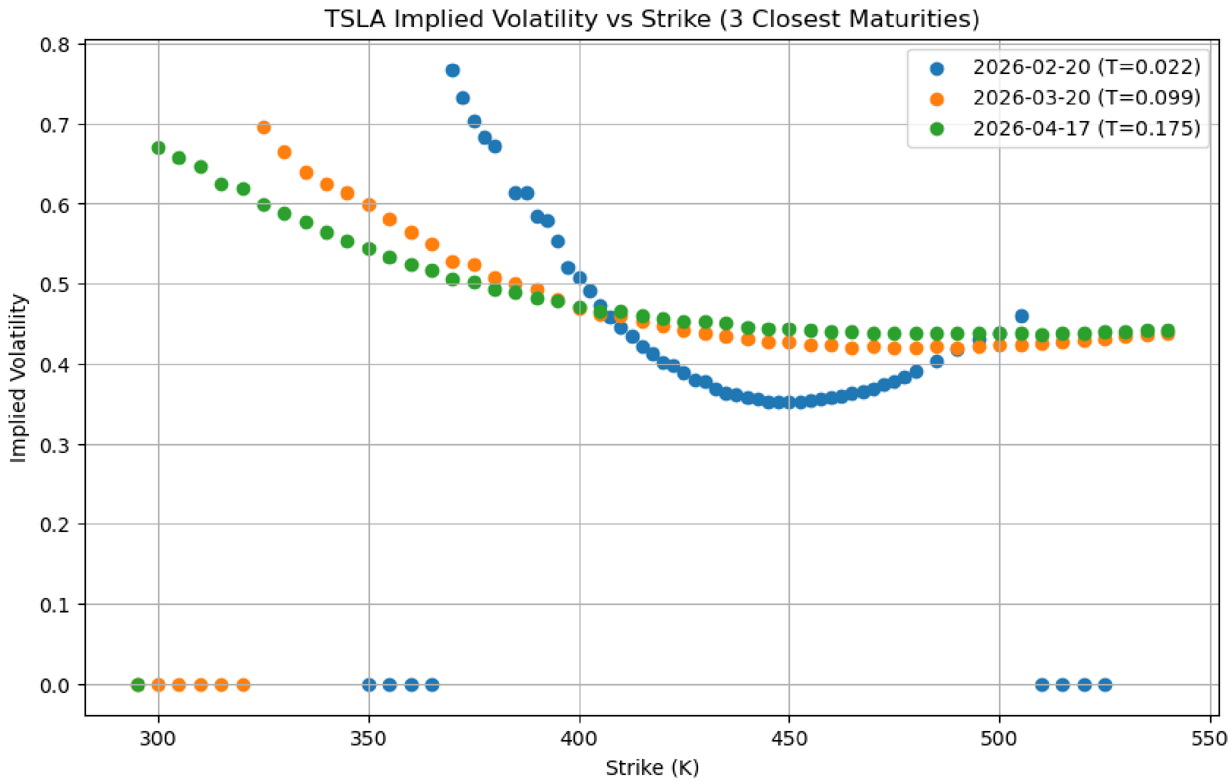
This calculation results in values:

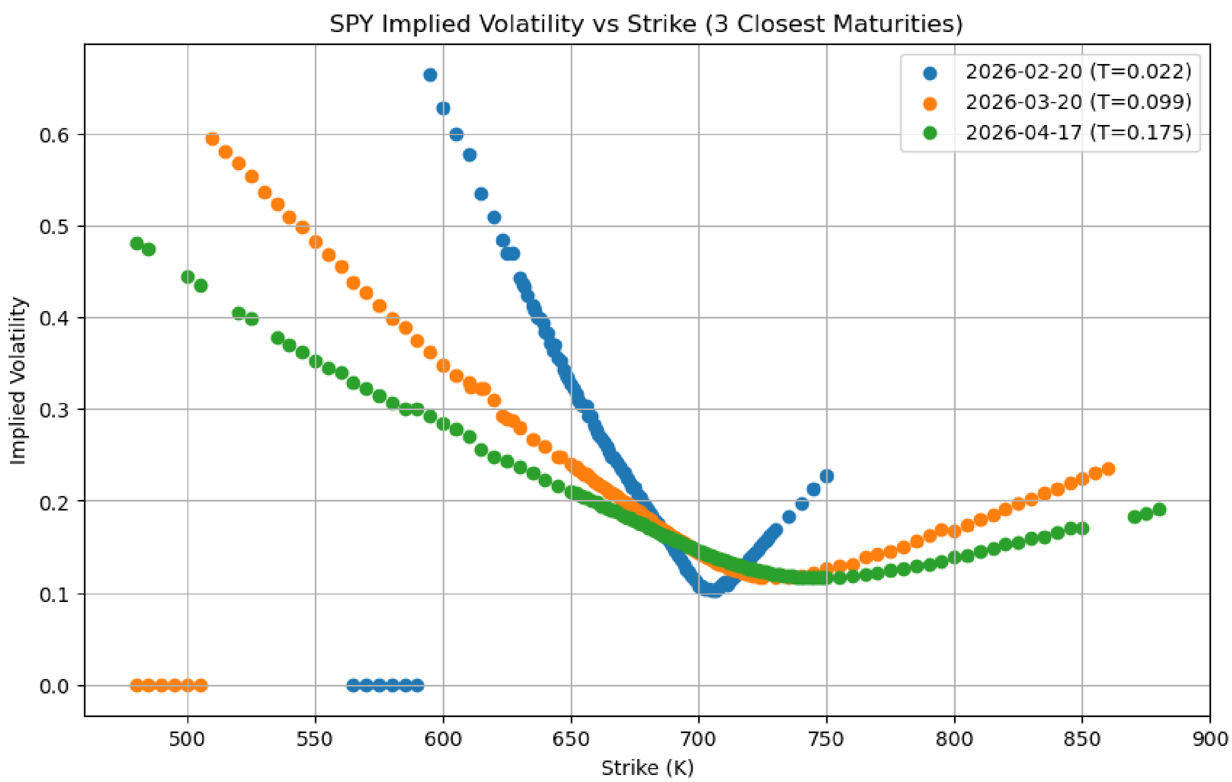
```
    Asset    Maturity  Strike  Call_Mid  Call_Calc  Call_Bid  Call_Ask  \
0    TSLA  2026-02-20   400.0    23.175  20.494139     23.05     23.30
1    TSLA  2026-02-20   402.5    21.100  18.451134     21.00     21.20
2    TSLA  2026-02-20   405.0    19.025  16.453128     18.95     19.10
3    TSLA  2026-02-20   407.5    17.075  14.480123     17.00     17.15
4    TSLA  2026-02-20   410.0    15.225  12.732118     15.15     15.30
..    ...         ...     ...       ...        ...       ...       ...
451   SPY  2026-04-17   696.0    12.370  12.313309     12.35     12.39
452   SPY  2026-04-17   697.0    11.890  11.774672     11.87     11.91
453   SPY  2026-04-17   698.0    11.380  11.046034     11.36     11.40
454   SPY  2026-04-17   699.0    10.945  10.777397     10.93     10.96
455   SPY  2026-04-17   700.0    10.405  10.343759     10.39     10.42

     Put_Mid   Put_Calc  Put_Bid  Put_Ask
0      2.670   5.350861     2.66     2.68
1      3.125   5.773866     3.10     3.15
2      3.625   6.196872     3.60     3.65
3      4.150   6.744877     4.10     4.20
4      4.900   7.392882     4.85     4.95
..       ...        ...      ...      ...
451   23.115  23.171691    22.84    23.39
452   23.570  23.685328    23.33    23.81
453   23.835  24.168966    23.71    23.96
454   24.560  24.727603    24.25    24.87
455   25.120  25.181241    24.92    25.32
```

Overall my calculated values for the SPY are very close to the mid prices, however my calculated values for TSLA seem to vary more.

2.10: I calculated the Implied volatiles for the 3 closest maturities. Those being (2/20, 3/20, 4/17). The plots look like:





As we can see, TSLA seems to have a very uniform smile to it while SPY seems to have a steeper downward slope. In terms of maturity, the later the maturity seems to have a more uniform shape with less volatility.

2.11: Using the B.S formula and Partials, I calculated the different Greeks. I bound the table from strike = 400 to strike = 700 due to certain Greeks being 1 or zero. With this constraint, SPY shows those being 0 or 1 since table is a small sample of the data. In the code it accounts for all values between the constraints. The table:

TSLA Greeks (400 ≤ K ≤ 700):

	Strike	IV	Delta_BSl	Delta_Numerical	Gamma_BS	Gamma_Numerical	Vega_BS	Vega_Numerical
0	400.0	0.468941	0.651302	0.651150	0.006016	0.006014	48.502686	48.502686
1	405.0	0.461618	0.620580	0.620444	0.006287	0.006286	49.903498	49.903498
2	410.0	0.459230	0.588145	0.588031	0.006463	0.006461	51.029426	51.029426
3	415.0	0.452734	0.555145	0.555053	0.006656	0.006653	51.810992	51.810992
4	420.0	0.446517	0.521228	0.521163	0.006804	0.006801	52.237468	52.237468

SPY Greeks (400 ≤ K ≤ 700):

	Strike	IV	Delta_BSl	Delta_Numerical	Gamma_BS	Gamma_Numerical	Vega_BS	Vega_Numerical
0	480.0	0.000001	1.0	1.0	0.0	0.0	0.0	1.012452e+06
1	485.0	0.000001	1.0	1.0	0.0	0.0	0.0	9.875414e+05
2	490.0	0.000001	1.0	1.0	0.0	0.0	0.0	9.626310e+05
3	495.0	0.000001	1.0	1.0	0.0	0.0	0.0	9.377206e+05
4	500.0	0.000001	1.0	1.0	0.0	0.0	0.0	9.128102e+05

```python
def bs_greeks(S, K, T, r, sigma):
    if T <= 0 or sigma <= 0:
        return np.nan, np.nan, np.nan

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

    delta = norm.cdf(d1)
    gamma = norm.pdf(d1) / (S * sigma * np.sqrt(T))
    vega  = S * norm.pdf(d1) * np.sqrt(T)

    return delta, gamma, vega
```

```python
def numerical_greeks(S, K, T, r, sigma):
    hS = 0.01 * S
    hV = 1e-4

    C0 = black_scholes_price(S, K, T, r, sigma, "call")

    C_up = black_scholes_price(S + hS, K, T, r, sigma, "call")
    C_down = black_scholes_price(S - hS, K, T, r, sigma, "call")

    delta_num = (C_up - C_down) / (2 * hS)
    gamma_num = (C_up - 2 * C0 + C_down) / (hS**2)

    C_vol_up = black_scholes_price(S, K, T, r, sigma + hV, "call")
    C_vol_down = black_scholes_price(S, K, T, r, sigma - hV, "call")

    vega_num = (C_vol_up - C_vol_down) / (2 * hV)

    return delta_num, gamma_num, vega_num
```

2.12: For this we calculated the price for the options in DATA2. The results are:

```
    Asset    Maturity  Strike  IV_from_DATA1  ModelPrice_DATA2  \
0    TSLA  2026-02-20   400.0       0.507961         22.500284
1    TSLA  2026-02-20   402.5       0.491663         20.420026
2    TSLA  2026-02-20   405.0       0.473250         18.345891
3    TSLA  2026-02-20   407.5       0.458152         16.394452
4    TSLA  2026-02-20   410.0       0.444687         14.544358
..    ...         ...     ...            ...               ...
412   SPY  2026-04-17   696.0       0.150381         12.215584
413   SPY  2026-04-17   697.0       0.149497         11.737658
414   SPY  2026-04-17   698.0       0.148250         11.230158
415   SPY  2026-04-17   699.0       0.147595         10.797164
416   SPY  2026-04-17   700.0       0.145881         10.260285

     MarketMid_DATA2
0             23.175
1             21.100
2             19.025
3             17.075
4             15.225
..               ...
412           12.370
413           11.890
414           11.380
415           10.945
416           10.405
```

As you can see, the calculated values from the data are very close to the market data. However they are not identical, meaning that based on my calculations there would be arbitrage opportunities.

Part 3: Numerical Integration of real-valued Functions. AMM arbitrage Fee Revenue

3.a:

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

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

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

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

By constraint, $(y_{t+1}x_{t+1}) = K$

Solving for X_t

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

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

->$\Delta X = \sqrt{\frac{k}{(S_{+}+1)(1-\gamma)}} - X_t$

Solving for y_t

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

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

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

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

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

-> $S_{t+1}/(1-\gamma) = (y_{t+1}/x_{t+1})$ or $(x_{t+1}/y_{t+1}) = (1-\gamma)/S_{t+1}$

Solving for x_t

-> $(x_{t+1})^2 * S_{t+1}/(1-\gamma) = K$

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

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

Solving for y_t

->$(y_{t+1})^2 * (1-\gamma)/S_{t+1} = K$

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

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

3.b:

Using the results and inputs provided, The estimation for E[R(St+1)] = 0.008496.

The code used for this was,

```python
def lognormal_pdf(s):
    mu = -0.5 * sigma**2 * dt
    var = sigma**2 * dt
    return (1 / (s * np.sqrt(2*np.pi*var))) * \
           np.exp(-(np.log(s) - mu)**2 / (2*var))
```

```python
upper_trigger = 1/(1-gamma)
lower_trigger = 1-gamma

s_max = 3
N = 10000
s_grid = np.linspace(1e-6, s_max, N)

integrand = np.zeros_like(s_grid)

#cases
for i, s in enumerate(s_grid):
    if s > upper_trigger:
        integrand[i] = gamma * delta_y(s) * lognormal_pdf(s)
    elif s < lower_trigger:
        integrand[i] = gamma * delta_x(s) * s * lognormal_pdf(s)
    else:
        integrand[i] = 0

expected_revenue = np.trapz(integrand, s_grid)
```

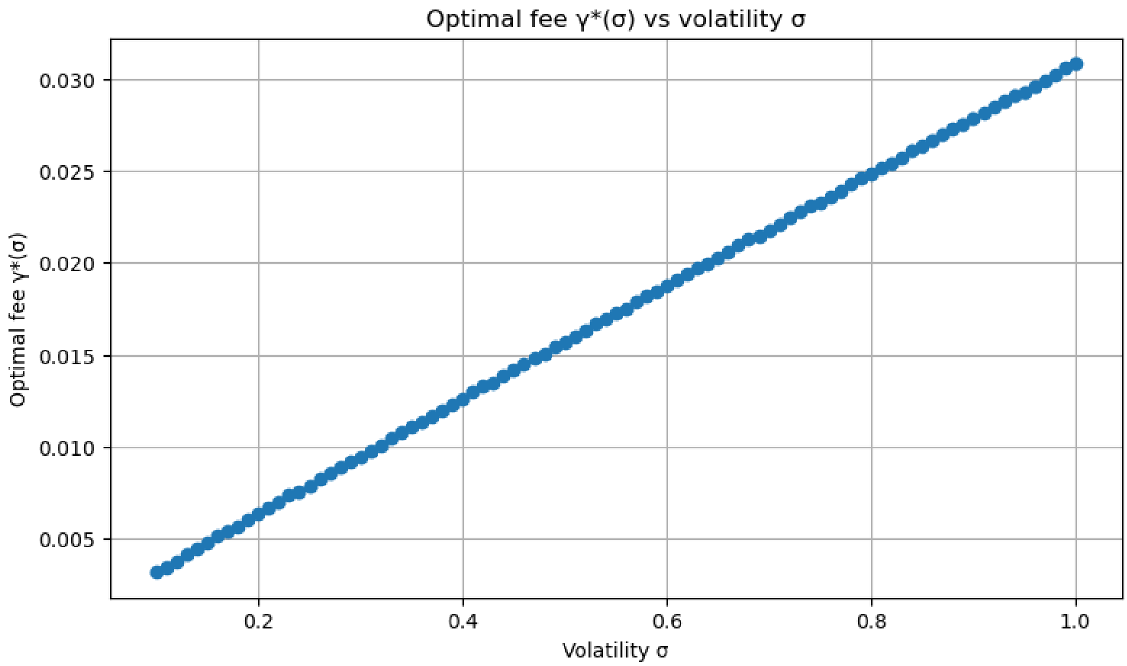
3.c:

Calculating for the given sigma and gamma, the results are:

sigma	gamma	ExpectedFeeRevenue
0.2	0.001	0.003682
0.2	0.003	0.008496
0.2	0.010	0.009337
0.6	0.001	0.011911
0.6	0.003	0.032884
0.6	0.010	0.080272
1.0	0.001	0.020041
1.0	0.003	0.057211
1.0	0.010	0.159083

The best is when gamma = .01 for each sigma as the expected fee revenue is the greatest.

When we calculate for the different sigma with step .001, we see that the optimal gamma* is the one correlating to the highest sigma. When we plot the points, we can see that this relationship follows a very linear relationship.


Optimal fee γ*(σ) vs volatility σ
Optimal fee γ*(σ)
Volatility σ
0.005
0.010
0.015
0.020
0.025
0.030
0.2
0.4
0.6
0.8
1.0

Both of these were used by a slight altercation between the code from part b.

```python
def expected_one_step_fee_revenue(sigma, gamma):
    upper_trigger = 1.0 / (1.0 - gamma)
    lower_trigger = 1.0 - gamma

    pdf_vals = lognormal_pdf(s_grid, sigma, dt)

    # piecewise integrand
    integrand = np.zeros_like(s_grid)

    mask_up = s_grid > upper_trigger
    mask_dn = s_grid < lower_trigger

    # R = 1{s>Pt/(1-g)} * g*delta_y + 1{s<Pt(1-g)} * g*delta_x*s
    integrand[mask_up] = gamma * delta_y(s_grid[mask_up], gamma) * pdf_vals[mask_up]
    integrand[mask_dn] = gamma * delta_x(s_grid[mask_dn], gamma) * s_grid[mask_dn] * pdf_vals[mask_dn]

    return np.trapz(integrand, s_grid)
```

```python
def expected_fee_revenue_given_pdf(pdf_vals, gamma):
    upper_trigger = 1.0 / (1.0 - gamma)
    lower_trigger = 1.0 - gamma

    integrand = np.zeros_like(s_grid)

    mask_up = s_grid > upper_trigger
    mask_dn = s_grid < lower_trigger

    # R = 1{s>1/(1-g)} * g*delta_y + 1{s<1-g} * g*delta_x*s
    integrand[mask_up] = gamma * delta_y_grid(gamma)[mask_up] * pdf_vals[mask_up]
    integrand[mask_dn] = gamma * delta_x_grid(gamma)[mask_dn] * s_grid[mask_dn] * pdf_vals[mask_dn]

    return np.trapz(integrand, s_grid)

sigmas = np.arange(0.10, 1.00 + 1e-12, 0.01)

# 0 to 10%
gamma_lo = 1e-6
gamma_hi = 0.10

gamma_star = []
```