Homework #1

Andre Sealy

February 15, 2026

Part 1: Data Gathering

The first part of the assignment involves downloading the following tickers: TSLA, SPY, and VIX. TSLA is the ticker for the stock Tesla, Inc.; SPY is the ticker for the State Street SPDR ETF Trust; and VIX is the Chicago Board Option Exchange (CBOE) Volatility Index. The SPY is a high liquid ETF that tracks the performance of the S&P 500 stock exchange. It's purpose is to provide investors access to the boarder market without the vast resources required to obtain positions in individual equities in the exchange. VIX is an index that tracks the expected volatility of the S&P 500, which are based on options on the S&P 500 index. It is often referred to as the "fear index."

The python program `1.1_download_options.py` find the third Friday of the end of each month (since options usually expire during this period) and extracts the calls and puts of options based on these expiration dates. This program is dynamic, which allows the user to extract most options in the market. However, options on VIX usually expires every Wednesday, so the python code `1.2_download_vix.py` is used to extract the calls and puts for these options. The value of a call under the Black-Scholes framework is denoted by $C_{BS}(S_0, K, T, r, \sigma)$, where S_0 is the stock price and r is the risk-free rate, which are known at the current time. We use the stock prices for Feburary 5th, 2026, which we record these values at 1:27 pm, Eastern Standard Time:

- **SPY**: 679.52
- **TSLA**: 399.63
- **VIX**: 20.54
- **r**: 3.62%

We use the risk-free rate on the day the option data and stock prices were recorded. We use the 3-month constant maturity Treasuries.

Also we have T which is the maturity in years, and σ is the implied volatility, which is unknown by the model but provided from the market. Data was downloaded from Yahoo! Finance using the `yfinance` library from python and exported to the file data1.csv. Although options traditionally expire on the third Friday of each month, the `yfinance` API will normally extract options with more frequent expiration date. However, in the case of TSLA, there are weekly options on the market that are expiring on March 6th, 13th, and 27th, in addition to March 20th, the third Friday of March. These options provide more flexibility for trading, such as short-term hedging and event trading, such as earnings announcements, macroeconomic releases, and Fed meetings. However, for the purpose of this assignment, we focus on call and put options from TSLA, SPY and VIX, which expire on the 20th of Feburary, March and April. The time to expiration, $T - t$, is denoted as the difference between the last trade date of the option and the expiration.

Part 2: Analyzing the Data

We first start off by constructing the analytical solution to the Black-Scholes formula, which is denoted by the following for both calls and puts

$$C(S,t) = SN(d_1) - Ke^{-r(T-t)}N(d_2) \quad P(S,t) = Ke^{-r(T-t)}N(d_2) - SN(d_1)$$

with d_1 and d_2 denoted as the following

$$d_1 = \frac{\log(S/K) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} \quad d_2 = d_1 - \sigma\sqrt{T-t}$$

and N is the Cumulative Distribution Function (CDF), denoted by the following

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}s^2} ds$$

Bisection Method

Since volatility cannot be directly observed, and because there is no closed-form expression for σ in Black-Scholes, we utilize a function that can numerically establish σ by utilizing the Bisection method. We define a function

$$f(\sigma) = C_{BS}(S_0, K, T, r, \sigma) - (B + A)/2$$

where the implied volatility is the root

$$\sigma = \{\sigma > 0 : f(\sigma) = 0\}$$

We choose $a > 0, b > 0$ such that $f(a)f(b) < 0$. Then we iterate on $m_n = (a_n + b_n)/2$ where B and A are the Bid and Ask prices of the option, respectively. We update the bracket by sign under the following conditions. If $f(a_n)f(m_n) \leq 0$, set $(a_{n+1}, b_{n+1}) = (a_n, m_n)$. Otherwise, we set $(a_{n+1}, b_{n+1}) = (m_n, b_n)$. We stop when $|f(m_n)| < \epsilon$ or $b_n - a_n < \epsilon$ and the output provides us with $\sigma \approx m_n$. The same idea is applied to puts.

The python program `2.1_bisection_method.py` allows the user to run the program by inputting the ticker, option type (call or put) and the stock price of the ticker. The program will read the csv from data1.csv and extract the necessary information for each option and numerically find the implied volatility. The program will output a table of the first 50 options with the ticker, option type, stock price, risk-free rate, as well as the implied volatility calculated from the numerical solution and the difference from of the volatility provided from Yahoo! Finance. The program will also save the results in a csv file. We have the results for TSLA calls TSLA calls and TSLA puts, SPY calls and SPY puts, VIX calls and VIX puts.

Newton Method

For the foot-finding problem for implied volatility, we use central-difference approximation with the following

$$f' \approx \frac{f(x+dx)-f(x-dx)}{2dx}, \quad dx = 10^{-5}$$

where the iteration is the following

$$\sigma_{n+1} = \sigma_n - \frac{f(\sigma_n)}{f'(\sigma_n)}, \quad f'(\sigma_n) \approx \frac{f(\sigma_n+dx)-f(\sigma_n-dx)}{2dx}$$

Similiar to bisection method, the program `2.2_newton_raphson_method.py` ask for the necessary inputs, such as the ticker, risk-free rate, option type and stock price. The program will numerically solve for σ for each option in the data1.csv, then output a table of the implied volatility and the difference from the volatility provided by Yahoo! Finance, then save the results in a csv file. We have the results for TSLA calls and TSLA puts, SPY calls and SPY puts, VIX calls and VIX puts.

Analysis of Numerical Methods

```
Ticker: TSLA
Option type: c
Stock Price (S): 399.63
Risk-Free Rate (r): 0.0362
Total options processed (after filters): 474

Comparison with market implied volatility:
Mean absolute difference: 0.1275250194502468

==================================================

Results
```

contractSymbol	strike	option_type	T	S	impliedVolatility	cal_implied_vol	vol_difference
TSLA260220C00100000	100.0	c	0.039078	399.63	3.162111	3.234005	0.071894
TSLA260220C00110000	110.0	c	0.285792	399.63	8.111577	3.152292	-4.959285
TSLA260220C00120000	120.0	c	0.093869	399.63	3.001956	1.933649	-1.068307
TSLA260220C00130000	130.0	c	0.093922	399.63	2.665042	1.659729	-1.005313
TSLA260220C00140000	140.0	c	0.093442	399.63	2.518558	1.567858	-0.950700
TSLA260220C00150000	150.0	c	0.039362	399.63	2.346684	2.370925	0.024242
TSLA260220C00155000	155.0	c	0.044385	399.63	2.291020	2.170002	-0.121018
TSLA260220C00160000	160.0	c	0.044501	399.63	2.211919	2.085927	-0.125992
TSLA260220C00165000	165.0	c	0.066792	399.63	2.109380	1.559804	-0.549576
TSLA260220C00170000	170.0	c	0.058242	399.63	2.035161	1.624596	-0.410565
TSLA260220C00175000	175.0	c	0.061242	399.63	2.152348	1.725794	-0.426554
TSLA260220C00180000	180.0	c	0.044415	399.63	2.008306	1.899663	-0.108643
TSLA260220C00185000	185.0	c	0.041984	399.63	2.099126	2.075873	-0.023253
TSLA260220C00190000	190.0	c	0.039089	399.63	1.858399	1.872449	0.014049
TSLA260220C00195000	195.0	c	0.063964	399.63	1.908692	1.485155	-0.423537

Figure 1: Example of TSLA Implied Volatilities using Bisection Method

Figure 1 shows an example of an output using the bisection method of TSLA options, alone with the average of the implied volatilities. We find that when implied volatility increases as $\sqrt{T-t}$ increases with maturity, which makes finding the root much easier with the bisection method. So at-the-money (ATM) options with longer expirations tend to have high implied volatility. However, as the data shows, extremely deep in-the-money (ITM) calls with small strike prices tend to have extremely high implied volatility. This can be explained as the solver tries to find σ such that $C_{BS}(\sigma) = C_{\text{market}}$, the intrinsic value becomes the value of the option. As such, volatility barely matters as any large σ will work numerically. Puts are the mirror opposite calls, where the explosion of volatility happens with deep ITM puts with large strike prices.

When solving numerically using the Newton-Raphson method, we find that root-finding tends to break down more often than the bisection method. The bisection method is slower; however, it's much more robust. The Newton method is fast, but it's fragile. As a result, the newton method fails to find the root and returns some missing values, which occurs where we know implied vol is small, such as deep ITM/OTM options, and short-dated expirations, while the newton method converges fastest near ATM options. On the other hand, when it comes to the SPY the newton method behaves much better.

Put-Call Parity

```
~/dev/FE621  master !3 ?6                                                   base  08:52:09 PM
❯ python 2.3_call_put_parity.py
ticker      contractSymbol option_type  strike  mid_price      S         T parity_price_type  parity_price
   SPY SPY260220C00335000           C   335.0    346.210 679.52 0.041068                 P      1.192341
   SPY SPY260220P00335000           P   335.0      0.005 679.52 0.041068                 C    345.022659
   SPY SPY260220P00340000           P   340.0      0.005 679.52 0.041068                 C    340.030086
   SPY SPY260220C00345000           C   345.0    336.120 679.52 0.041068                 P      1.087486
   SPY SPY260220P00345000           P   345.0      0.005 679.52 0.041068                 C    335.037514
   SPY SPY260220C00350000           C   350.0    331.615 679.52 0.041068                 P      1.575058
   SPY SPY260220P00350000           P   350.0      0.005 679.52 0.041068                 C    330.044942
   SPY SPY260220P00355000           P   355.0      0.005 679.52 0.041068                 C    325.052370
   SPY SPY260220P00360000           P   360.0      0.005 679.52 0.041068                 C    320.059797
   SPY SPY260220C00365000           C   365.0    326.975 679.52 0.041068                 P     11.912775
   SPY SPY260220P00365000           P   365.0      0.005 679.52 0.041068                 C    315.067225
   SPY SPY260220C00370000           C   370.0    322.060 679.52 0.041068                 P     11.990347
   SPY SPY260220P00370000           P   370.0      0.005 679.52 0.041068                 C    310.074653
   SPY SPY260220C00375000           C   375.0    306.375 679.52 0.041068                 P      1.297919
   SPY SPY260220P00375000           P   375.0      0.015 679.52 0.041068                 C    305.092081
   SPY SPY260220C00380000           C   380.0    301.435 679.52 0.041068                 P      1.350492
```

Figure 2: Call-Put parity calculation

Figure 2 demonstrates is the python output for program `2.3_call_put_parity.py` with the respective call and put pairs based on their strike price. Recall that call-put parity has the following relationship

$$C_t + Ke^{-r(T-t)} = P_t + S_t$$

We can see when we incorporate the mid price (the average between the bid and ask), the parity holds for the most part.

Implied Volatility Plot

The python program `2.4_iv_plots.py` allows the user to plot the different expiration of options based on the ticker, option type, and the numerical method.Figure ?? shows the implied volatility plot in percentage. Overall, the plot shows an asystemical smile, with very high implied volatility with low strikes (deep ITM calls). Volatility falls as the price reaches towards ATM point and raises again for higher strike prices (OTM calls). For shorter maturities, (2026-02-20) we see that implied volatility is also genearlly lower and flatter, indicating more of a smirk than a smile. For longer maturities (2026-04-17), we see that volatility is the greatest, where the smirk is the strongest.

These facts suggest that high volatility on low strikes suggests markets assign more probability to downside moves (crash risk), which is typical for equities. For deep ITM calls, volatility is likely driven by illiquidity and stale quotes rather than true beliefs about volatility.

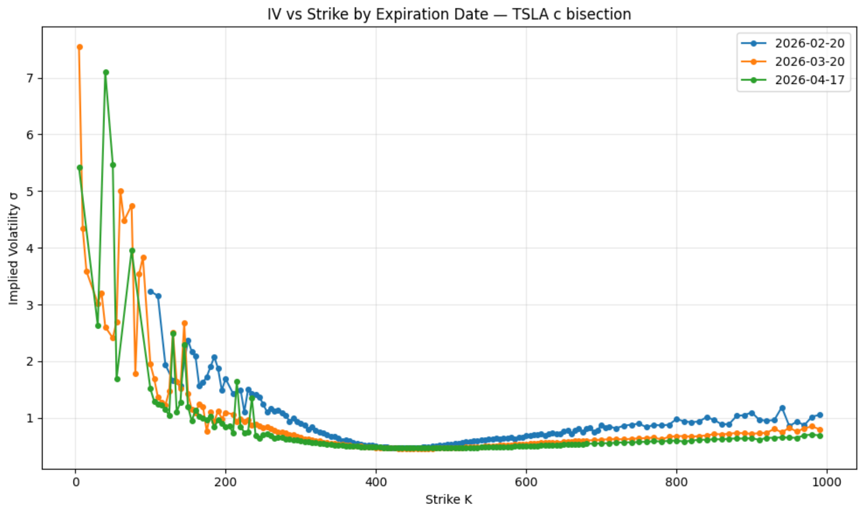

Figure 3: Implied Volatility plots

Greeks

For this section, we calculate the volatility greeks: delta Δ, gamma Γ, and *Vega*, which are denoted as the following

$$\Delta = \frac{\partial C}{\partial S} \quad \Gamma = \frac{\partial^2 C}{\partial S^2} \quad Vega = \frac{\partial C}{\partial \sigma}$$

We can use a closed-form solution for calculating the greeks, however, we can also use a numerical solution for approximating partial differential equations, known as the finite difference method. The finite difference method approximates the differential operator by replacing the derivatives in the equation using differential quotients. We have three different types of differntial quotients, the first being the forward difference

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} + O(\Delta x),$$

the backward difference

$$f'(x) = \frac{f(x - \Delta x) - f(x)}{\Delta x} + O(\Delta x),$$

and the central difference

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} + O(\Delta x)^2,$$

Analytically, the Delta Δ simplies to the CDF evaluated at $\pm d_1$,

$$\frac{\partial C}{\partial S} = N(d_1) \quad \frac{\partial P}{\partial S} = -N(-d_1),$$

however, the delta finite difference is the following

$$\frac{\partial C}{\partial S} = \frac{BS_{\text{Call}}(S + \Delta S, K, T, r, \sigma) - BS_{\text{Call}}(S, K, T, r, \sigma)}{\Delta S}.$$

The analytical solution for Γ is the following

$$\frac{\partial^2 C}{\partial S^2} = \frac{\partial^2 P}{\partial S^2} = \frac{N'(d_1)}{S\sigma\sqrt{T}},$$

but the finite difference for gamma is as follows

$$\frac{\partial P}{\partial S} = \frac{BS_{\text{Put}}(S + \Delta S, K, T, r, \sigma) - 2BS_{\text{Put}}(S, K, T, r, \sigma) + BS_{\text{Put}}(S - \Delta, K, T, r, \sigma)}{(\Delta S)^2}.$$

Similar to Gamma, *Vega* analytical solution for calls and puts are the same, so we have

$$\frac{\partial C}{\partial \sigma} = \frac{\partial P}{\partial \sigma} = \frac{N'(d_1)}{S\sigma\sqrt{T}},$$

which gives as the finite difference of the following

$$\frac{\partial C}{\partial \sigma} = \frac{BS_{\text{Call}}(S + \Delta S, K, T, r, \sigma) - BS_{\text{Call}}(S, K, T, r, \sigma)}{\Delta S}.$$

The python program `2.5_greeks.py` calculates the deltas, gammas, and vegas for all of the options in the csv file data1.csv, based on the option type. It also calculates the same greeks using the finite difference method. All greeks calculated using this method are calculated using the central method. The output can be found in the greeks.csv. Greeks calculated via Black-Scholes are denoted by bs; while greeks calculate via finite difference method are denote via fdm. When compared, the finite difference method is accurate to the one-hundred Billionth.

Predicting Day 2 Options

Now we have the task of using the implied volatilities that we have calculated and using them to predict the option prices for the next day. For that, we use the stock prices for February 6th, 2026

```
Loaded: data/TSLA_c_bisection.csv
Valuation date: 2026-02-06 00:00:00+00:00
Rows after option_type filter: 470
     contractSymbol ot  strike       T    S_2  cal_implied_vol   bs_price  optionPrice  price_dff_day2
TSLA260220C00100000  c   100.0 0.03833 410.45         3.234005 311.135474       297.06       14.075474
TSLA260220C00110000  c   110.0 0.03833 410.45         3.152292 301.339534       321.40      -20.060466
TSLA260220C00120000  c   120.0 0.03833 410.45         1.933649 290.627159       319.63      -29.002841
TSLA260220C00130000  c   130.0 0.03833 410.45         1.659729 280.631924       309.87      -29.238076
TSLA260220C00140000  c   140.0 0.03833 410.45         1.567858 270.646090       299.78      -29.133910
TSLA260220C00150000  c   150.0 0.03833 410.45         2.370925 261.252335       245.82       15.432335
TSLA260220C00155000  c   155.0 0.03833 410.45         2.170002 256.052681       262.16       -6.107319
TSLA260220C00160000  c   160.0 0.03833 410.45         2.085927 251.034771       258.53       -7.495229
TSLA260220C00165000  c   165.0 0.03833 410.45         1.559804 245.707528       280.19      -34.482472
TSLA260220C00170000  c   170.0 0.03833 410.45         1.624596 240.751703       232.23        8.521703
TSLA260220C00175000  c   175.0 0.03833 410.45         1.725794 235.854710       262.03      -26.175290
TSLA260220C00180000  c   180.0 0.03833 410.45         1.899663 231.156615       239.00       -7.843385
TSLA260220C00185000  c   185.0 0.03833 410.45         2.075873 226.732356       210.42       16.312356
TSLA260220C00190000  c   190.0 0.03833 410.45         1.872449 221.354831       208.30       13.054831
TSLA260220C00195000  c   195.0 0.03833 410.45         1.485155 215.850930       206.99        8.860930
TSLA260220C00200000  c   200.0 0.03833 410.45         1.690697 211.214414       210.08        1.134414
TSLA260220C00210000  c   210.0 0.03833 410.45         1.429608 200.960076       188.76       12.200076
TSLA260220C00220000  c   220.0 0.03833 410.45         1.493760 191.259716       189.36        1.899716
```

Figure 4:

- **SPY**: 687.47
- **TSLA**: 410.45
- **VIX**: 18.03
- **r**: 3.58%

The prediction runs with the use of python program `2.6_predict_option.py`, which requires the user to input the ticker, option type, approximation method and risk-free rate. The program will used the saved csv files to collect data, which will be used to determine the next day option prices based on the implied volatility calculated for each option, which was derived via bisection or Newton-Raphson method. Figure

Problem 3

For the

Answer a)

To derive the swap amounts, we derive the arbitrage trade size that move the Automated MarketMaker (AMM) price to the boundary of the no-arbitrage region after the external price change from S_t to S_{t+1}. At time t the AMM reserve are x_t (BTC) and y_t (USDC), with constant-product invariant $x_t y_y = k$. The pool mid-price is,

$$P_t = \tfrac{y_t}{x_t} \tag{1}$$

Let $\gamma \in (0, 1)$ denote the swap fee parameter. Only the fraction $(1 - \gamma)$ for the input asset enters the pool.

After the external price moves to S_{t+1}, arbitraguers trade until the pool price reaches the boundary of the no-arbitrage boundary, we are derived from the following cases.

Case 2

When BTC is cheaper in the pool,

$$S_{t+1} > \frac{P_t}{1-\gamma}$$

arbitgraguers will want to swap USDC for BTC. Let the trade size be $\Delta x > 0$, where BTC is out of the pool and $\Delta y > 0$, where USDC is in the pool. After the trade we have

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

Then the following the invariant must hold: $x_{t+1}y_{t+1} = k$. The arbitrage pool price must be equal to the upper no-arbitrage boundary, such that

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

Then by equation 1, we have the following boundary condition

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

From equation 2, we solve the system of linear equations,

$$\begin{aligned} x_{t+1}\left((1-\gamma)S_{t+1}x_{t+1}\right) &= k \\ (1-\gamma)S_{t+1}x_{t+1}^2 &= k \end{aligned}$$

Solving for both x_t and y_t, we have the following:

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

Therefore, the swap amounts are the following

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

Case 1

When BTC is cheaper outside the pool,

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

where the arbigraguers will want to swap BTC for USDC. In this scenario, we swap BTC for USDC, such that

$$\begin{aligned} x_{t+1} &= x_t + (1-\gamma)\Delta x \\ y_{t+1} &= y_t - \Delta y \end{aligned}$$

After the arbitrage the pool price equals the lower boundary, $P_{t+1}(1-\gamma) = S_{t+1}$, therefore,

$$\frac{y_{t+1}}{x_{t+1}} = \frac{S_{t+1}}{1-\gamma} \tag{3}$$

So using the following,

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

we solve the following system of linear equation,

$$\begin{aligned} x_{t+1}\frac{S_{t+1}}{1-\gamma} &= k \\ \frac{S_{t+1}}{1-\gamma}x_{t+1}^2 &= k \end{aligned}$$

Solving for x_t and y_t, we have the following,

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

with the following swap amounts,

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

Thus we have the one-step revenue

$$R(S_{t+1}) = \mathbb{1}_{\{S_{t+1} > P_t/1\,-\,\gamma\}}\gamma\Delta y + \mathbb{1}_{\{S_{t+1} < P_t(1-\gamma)\}}\gamma\Delta x S_{t+1} \tag{4}$$

Answer b)

Given $P_t = \frac{y_t}{x_t} = 1$, and $k = x_t y_t = 10^6$. The no-arbitrage band is the following

$$S_{t+1} \in [P_t(1-\gamma), P_t/(1-\gamma)] = [1-\gamma, 1/(1-\gamma)] .$$

For cases when we swap USDC for BTC, we have $S_{t+1} > 1/(1-\gamma)$, which makes the fee $\gamma\Delta y(S_{t+1})$, we have the following

$$\Delta y(s) = \frac{1}{1-\gamma}\left(\sqrt{k(1-\gamma)s} - y_t\right).$$

and when we swap BTC for USDC, for which $0 < S_{t+1} < 1-\gamma$, we have a fee of $\gamma\Delta x(S_{t+1}S_{t+1})$, which gives us the following

$$\Delta x(s) = \frac{1}{1-\gamma}\left(\sqrt{\frac{k(1-\gamma)}{s}} - x_t\right)$$

so the intergrand for the following

$$g_1(s) = \gamma\Delta y(s) f_{S_{t+1}}(s), \quad g_2(s) = \gamma\Delta x(s) s f_{S_{t+1}}(s)$$

Then the expectation is the following

$$\mathbb{E}\left[R(S_{t+1})\right] = \int_{1/(1-\gamma)}^{\infty} g_1(s)ds + \int_0^{1-\gamma} g_2(s)ds \tag{5}$$

Now we have the Geometric Brownian Motion

$$S_{t+1} = S_t \exp\left(-\frac{1}{2}\sigma^2\Delta + \sigma\sqrt{\Delta t}Z\right), \quad Z \sim \mathcal{N}(0,1), \quad S_t = 1$$

with the lognormal density

$$\ln S_{t+1} \sim \mathcal{N}\left(\mu, v^2\right), \quad \mu = -\frac{1}{2}\sigma^2\Delta t, = \sigma\sqrt{\Delta t}$$

Hence for $s > 0$, we have the following

$$f_{S_{t+1}}(s) = \frac{1}{sv\sqrt{2\pi}}\exp\left(-\frac{(\ln s - \mu)^2}{2v^2}\right)$$

With equation 5, we can't numerically integrate to ∞, so we approximate the upper tail integral by integrating up to a large $s_{\max} = \exp(\mu + mv)$, where $m = 6$. This will help us capture most of the probability mass function. The lower half of the integral, we let > 0 and $s_{\min} = \epsilon$. Using the trapezoidal rule for the interval $[s_{\min}, 1-\gamma]$, we choose N_2 subintervals, which gives us the following step size

$$h_2 = \frac{(1-\gamma) - s_{\max}}{N_2}, \quad s_i = s_{\min} + ih_2$$

so the trapezoid approximation is the following

$$\int_{s_{\min}}^{1-\gamma} g_2(s)ds \approx h_2\left[\frac{g_2(s_0) + g_2(s_{N_2})}{2} + \sum_{i=1}^{N_2-1} g_2(s_i)\right]$$

For the interval $[1/(1-\gamma), s_{\max}]$, we have the following step size

$$h_1 = \frac{s_{\max} - \frac{1}{1-\gamma}}{N_1}, \quad u_j = \frac{1}{1-\gamma} + jh_1.$$

So the Trapezoid approximation is the followiong

$$\int_{1/(1-\gamma)}^{s_{\max}} g_1(s)ds \approx h_1\left[\frac{g_1(u_0 + g_1(u_{N_1}))}{2} + \sum_{j=1}^{N_1-1} g_1(u_j)\right]$$

So the formal trapezoidal method for the numerical approximation is the following

$$\mathbb{E}\left[R(S_{t+1})\right] \approx h_2\left[\frac{g_2(s_0) + g_2(S_{N_2})}{2} + \sum_{i=1}^{N_2-1} g_2(s_i)\right] + h_1\left[\frac{g_1(u_0) + g_1(u_{N_1})}{2} + \sum_{j=1}^{N_1-1} g_1(u_j)\right]$$

Assuming the initial BTC/UTC pool reserves are $x_t = y_t = 1000, P_t = \frac{y_t}{x_t} = 1, S_t = 1, \Delta t = \frac{1}{365}, \sigma = 0.2$, and $\gamma = 0.003$, we have an expected one-day revenue fee of $\mathbb{E}\left[R(S_{t+1})\right] \approx 0.008522\ldots$

Answer c)

Recall from equation 4, for each (σ, γ) we have the following:

$$\begin{aligned}
\mathbb{E}\left[R(S_{t+1})\right] &= \int_0^{\infty} \left[\mathbb{1}_{\{S_{t+1} > P_t/1\,-\,\gamma\}} \gamma \Delta y(s) + \mathbb{1}_{\{S_{t+1} < P_t(1-\gamma)\}} \gamma \Delta x(s) s\right] f_{S_{t+1}}(s) ds \\
&= \int_0^{P_t(1-\gamma)} \gamma \Delta x(s) s f_{S_{t+1}}(s) ds + \int_{P_t/(1-\gamma)}^{\infty} \gamma \Delta y(s) f_{S_{t+1}}(s) ds
\end{aligned}$$

Where $f_{S_{t+1}}$ is the lognormal denity implied by the one-step Geometric Brownian Motion. The swap functions, $\Delta x, \Delta y$, becomes the following

- If $s < P_t(1-\gamma)$,

$$\Delta x(s;\gamma) = \frac{1}{1-\gamma}\left(\sqrt{\frac{k(1-\gamma)}{s}} - x_t\right)$$

- If $s < \frac{P_t}{1-\gamma}$,

$$\Delta y(s;\gamma) = \frac{1}{1-\gamma}\left(\sqrt{k(1-\gamma)s} - y_t\right)$$

Given the sets $\sigma = \{0.2, 0.6, 1.0\}$, and $\gamma = \{0.001, 0.003, 0.01\}$, we compute $\mathbb{E}\left[R(S_{t+1})\right]$ for all 9, then for each σ the following

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

For each $\sigma \in [0.1, 1.0]$, we compute $\mathbb{E}\left[R(S_{t+1})\right], \forall \gamma = \{0.001, 0.003, 0.01\}$. We set $\gamma^*(\sigma)$ to the maximizer and then plot σ along with $\gamma^*(\sigma)$.

σ	$\gamma = 0.001$	$\gamma = 0.003$	$\gamma = 0.01$
0.2	0.003685	0.008522	0.009430
0.6	0.011923	0.032983	0.081082
1.0	0.020061	0.057384	0.160690

Table 1: Expected one-step fee revenue $\mathbb{E}[R]$

Table 1 shows the relationship between the volatility σ and the fee rate γ. For any fixed fee rate, expected revenue increases sharply with volatility. Higher volatility means the external

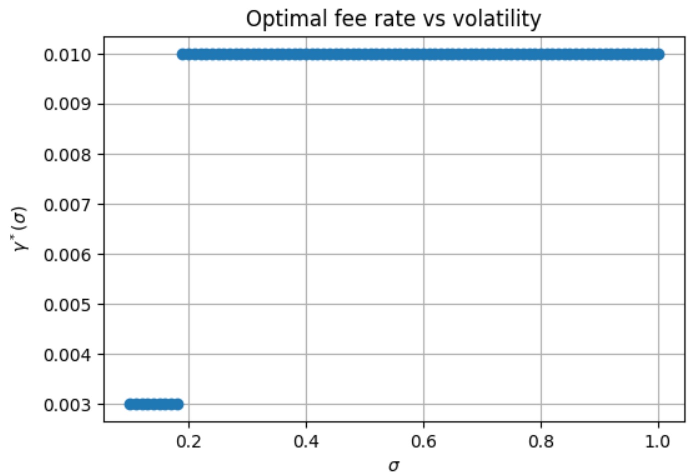

Figure 5: Optimal Fee rate

price moves more in a single day. Large price movements pushes the pool price outside the no-arbitrage bound more often. When this happens, an arbitrage trader rebalances the pool and fee revenue increases. This allows AMMs to earn more in volatility markets, which is volatility harvesting. For any fixed volatility, revenue increases as the fee rate rises, which also widens the no-arbitrage bound range, which reduces the arbitrage frequency. Using this table, we find that the optimal fee rate $\gamma^*(\sigma) = 0.01$.

Figure 4 shows how the optimal fee rate increases for any given σ. As we can see, when volatility increases the optimal fee rate increases when $\gamma = 0.01$