Homework 1
FE 621 – Methods in Computational Finance

Justin Zachery

February 2026

Contents

1 Section 1: Data Gathering Component (Questions 1–4)

1.1 Question 1: Data Download Program

I utilized R programming language for this assignment and pulled data from Yahoo Finance with the library quantmod in R. Please see Appendix Section 1 to see the code for this, as well as the BONUS part, which is downloading multiple assets and saving them in a single CSV file.

1.2 Question 2: Downloaded Assets and Data Sets

TSLA, SPY, and ˆVIX data were downloaded for two consecutive trading days. The first day's data set is referred to as DATA1, which was January 29th, 2026, and the second day's data set is referred to as DATA2, which is January 30th, 2026. The code implementation is provided in Appendix Section 1.

The reason there are so many maturities is that they allow for more interaction with the market, which, in turn, is beneficial for the overall market. Having more interaction with the market allows for more accurate volatility, making it easier to manage risk.

1.3 Question 3: Description of Assets

This section explains TSLA, SPY (ETF), and ˆVIX and their respective market purposes. There is no coding component for this.

TSLA: Tesla, Inc. common stock traded on the U.S. equity market. It represents ownership in Tesla and is used as the underlying asset for its listed equity options.

SPY: The SPDR S&P 500 ETF Trust. This exchange-traded fund tracks the performance of the S&P 500 Index, which represents approximately 500 large-cap U.S. companies. It is commonly used as a proxy for overall U.S. equity market performance.

ˆVIX: The CBOE Volatility Index (VIX). It measures the market's expectation of 30-day forward-looking volatility implied by S&P 500 index options. It is commonly interpreted as a measure of market uncertainty or risk sentiment.

The variance used in the VIX calculation is given by:

$$\sigma^2 = \frac{2}{T}\sum_i \frac{\Delta K_i}{K_i^2} e^{RT} Q(K_i) - \frac{1}{T}\left(\frac{F}{K_0} - 1\right)^2 \tag{1}$$

The VIX index level is then computed as:

$$\text{VIX} = 100 \times \sqrt{\sigma^2} \tag{2}$$

where σ^2 represents the 30-day forward variance, T is time to expiration (in years), F is the forward index level, K_i are the strike prices, K_0 is the first strike below the forward level, R is the risk-free interest rate, $Q(K_i)$ is the midpoint option quote, and ΔK_i is half the difference between adjacent strike prices.

To provide information on option expiration, I used the dates February 20th, March 20th, and April 17th for TSLA and SPY. For VIX, I used the expirations on February 18th, March 18th, and April 15th. For these 3 entities I used the year 2026.

1.4 Question 4: Recorded Market Information

This section discusses the recorded underlying prices, short-term rate selection and conversion, and time-to-maturity calculations. The code implementation is provided in the Appendix Section 1.

Table 1: Recorded Underlying Prices at Time of Data Download

Asset	Price (S_0)
TSLA	415.6069
SPY	684.4450
ˆVIX	19.56

The short-term risk-free interest rate was obtained from the website provided in the assignment handout. The reported annualized rate was $r = 0.0364$ (3.64%) for both January 29 and January 30.

The time to maturity was calculated as:

$$T = \frac{\text{Expiration Date} - \text{Valuation Date}}{365} \tag{3}$$

where T represents time to maturity in years.

Table 2: Time to Maturity (in Years)

Maturity	T (Years)
February	0.0411
March	0.1178
April	0.1945

2 Section 2: Analysis of the Data (Questions 5–12)

2.1 Question 5: Black–Scholes Pricing Model

The code implementation is provided in the Appendix Section 2.

2.2 Question 6: Implied Volatility via Bisection Method

The code implementation is provided in the Appendix Section 2.

Table 3: Implied Volatility Results (Bisection Method)

Symbol	S_0	τ	K_{ATM}	IV Call	IV Put	Avg IV (ATM)	Avg IV (Band)	Exp
"TSLA"	415.6069	0.0602739726027397	415	0.253514330182448	0.252685565015502	0.253099947598975	0.254073950164078	"Feb"
"TSLA"	415.6069	0.136986301369863	415	0.373682485422418	0.371977420393825	0.372829952908121	0.373069365626188	"Mar"
"TSLA"	415.6069	0.213698630136986	415	0.412212708238527	0.40921390602738	0.410713307132954	0.409591169592913	"Apr"
"SPY"	684.445	0.0602739726027397	684	0.102693449415997	0.100874502285756	0.101783975850876	0.104900513138873	"Feb"
"SPY"	684.445	0.136986301369863	684	0.140157214965206	0.147016534740545	0.143586874852875	0.142268640382177	"Mar"
"SPY"	684.445	0.213698630136986	684	0.143564681440361	0.155954882736295	0.149759782088328	0.148956898079516	"Apr"

From looking at the table (Avg IV ATM) represents the average implied volatilty and the (Avg IV Band) is volatilities that correspond to the moneyness band of 0.95 - 1.05.

2.3 Question 7: Newton/Secant Method

The code implementation is provided in the Appendix Section 2.

Table 4: Implied Volatility Results (Newton Method)

Symbol	S_0	τ	K_{ATM}	IV Call	IV Put	Avg IV (ATM)	Avg IV (Band)	Exp
"TSLA"	415.6069	0.0602739726027397	415	0.253514328811013	0.252685568097168	0.253099948454091	0.254073952397063	"Feb"
"TSLA"	415.6069	0.136986301369863	415	0.373682486022191	0.371977423445579	0.372829954733885	0.373069365041715	"Mar"
"TSLA"	415.6069	0.213698630136986	415	0.412212717707192	0.409213909899751	0.410713313803472	0.409591172695303	"Apr"
"SPY"	684.445	0.0602739726027397	684	0.102693453311981	0.100874501431872	0.101783977371926	0.104900519862958	"Feb"
"SPY"	684.445	0.136986301369863	684	0.140157221859893	0.147016529009678	0.143586875434786	0.142268640863372	"Mar"
"SPY"	684.445	0.213698630136986	684	0.143564686691922	0.155954881094711	0.149759783893317	0.148956898963527	"Apr"

Table 5: Convergence Time Comparison (Milliseconds)

Method	Min	LQ	Mean	Median	UQ	Max	Runs
"Bisection"	0.14454	0.1473915	0.15572129	0.1494105	0.151711	0.244599	100
"Newton"	0.030241	0.0317955	0.03730057	0.033184	0.0364805	0.081495	100

From looking at the table, it is apparent that Newton's method has a more efficient convergence time than bisection. This makes sense because Newton's method is a quadratic convergence, while bisection is a linear convergence.

2.4 Question 8: Implied Volatility Table and Discussion

The code implementation is provided in the Appendix Section 2.

I will be using the implied volatilities calculated from Q6 with the bisection method.

Table 6: Implied Volatility Summary by Maturity (Q8)

Symbol	Exp	τ	K_{ATM}	IV Call	IV Put	Avg IV (ATM)	Avg IV (Band)
"SPY"	"Apr"	0.213699	684	0.143565	0.155955	0.14976	0.148957
"SPY"	"Feb"	0.060274	684	0.102693	0.100875	0.101784	0.104901
"SPY"	"Mar"	0.136986	684	0.140157	0.147017	0.143587	0.142269
"TSLA"	"Apr"	0.213699	415	0.412213	0.409214	0.410713	0.409591
"TSLA"	"Feb"	0.060274	415	0.253514	0.252686	0.2531	0.254074
"TSLA"	"Mar"	0.136986	415	0.373682	0.371977	0.37283	0.373069

From looking at the implied volatilities of SPY and TSLA it is apparent that TSLA has a higher implied volatility than SPY this would make sense since SPY represents a group of stocks, while TSLA is a singular stock.

The current value of VIX is 21.09. Which is slightly higher than the IV calculated for SPY. This is because the VIX represents a 30-day period, so this could indicate that volatility will rise in the coming month for SPY. From the table, looking at volatility as maturity increases, it shows that as you increase the maturity time, the volatility increases. This is because there is more uncertainty farther out in the market.

When looking at options that are out of the money and in the money, it is apparent that as an option becomes more in the money, the volatility decreases.

2.5 Question 9: Put–Call Parity

The code implementation is provided in the Appendix Section 2.

When looking at the option prices these are for stocks that are At The Money (ATM), so there are some options that fall above this with the bid and ask and some fall below this for the bid and ask. The farther the stock is above this price, the more Out of The Money (OTM) the option is, and if it is below this price, then it is considered In The Money (ITM).

Table 7: ATM Option Prices Used for Put–Call Parity (Q9)

Symbol	Type	Price	Month
"TSLA"	"Call"	11.0414	"Feb"
"TSLA"	"Put"	9.5586	"Feb"
"TSLA"	"Call"	24.0961	"Mar"
"TSLA"	"Put"	21.5289	"Mar"
"TSLA"	"Call"	33.1225	"Apr"
"TSLA"	"Put"	29.5275	"Apr"
"SPY"	"Call"	7.769	"Feb"
"SPY"	"Put"	5.946	"Feb"
"SPY"	"Call"	16.8171	"Mar"
"SPY"	"Put"	12.2829	"Mar"
"SPY"	"Call"	22.6049	"Apr"
"SPY"	"Put"	15.3151	"Apr"

2.6 Question 10: Volatility Smile Plots

The code implementation is provided in the Appendix Section 2. As well as the BONUS part which is the 3D plots.

From looking at the plots below as you increase the maturity, the curve gets less sharp or flatter. Meaning the curve is less pronounced. This is because it is looking at a window that is larger so short term jumps in the market have less of an affect.

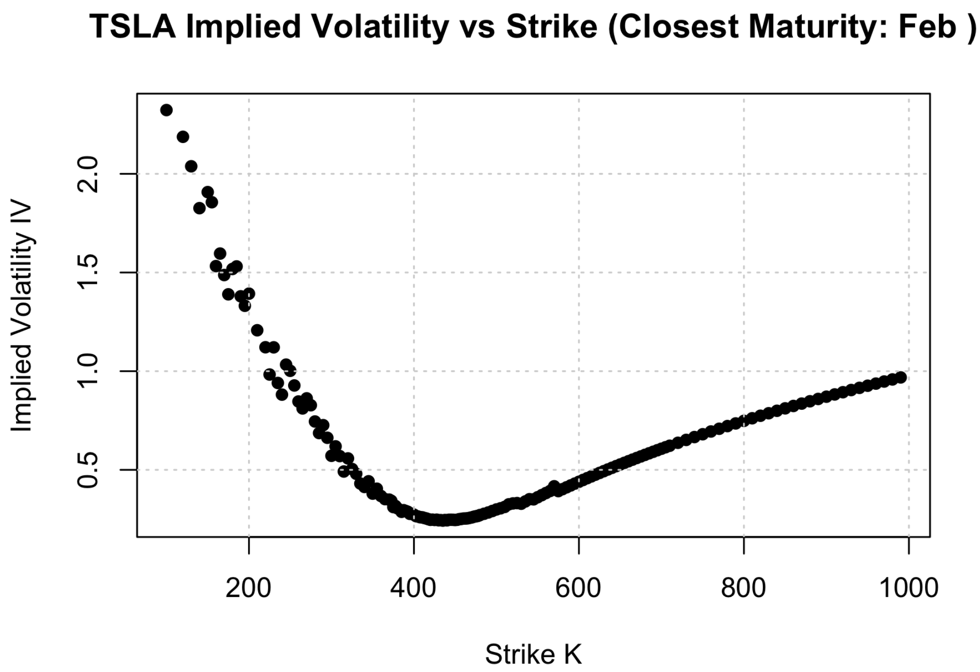

Figure 1: TSLA Implied Volatility Plot 1

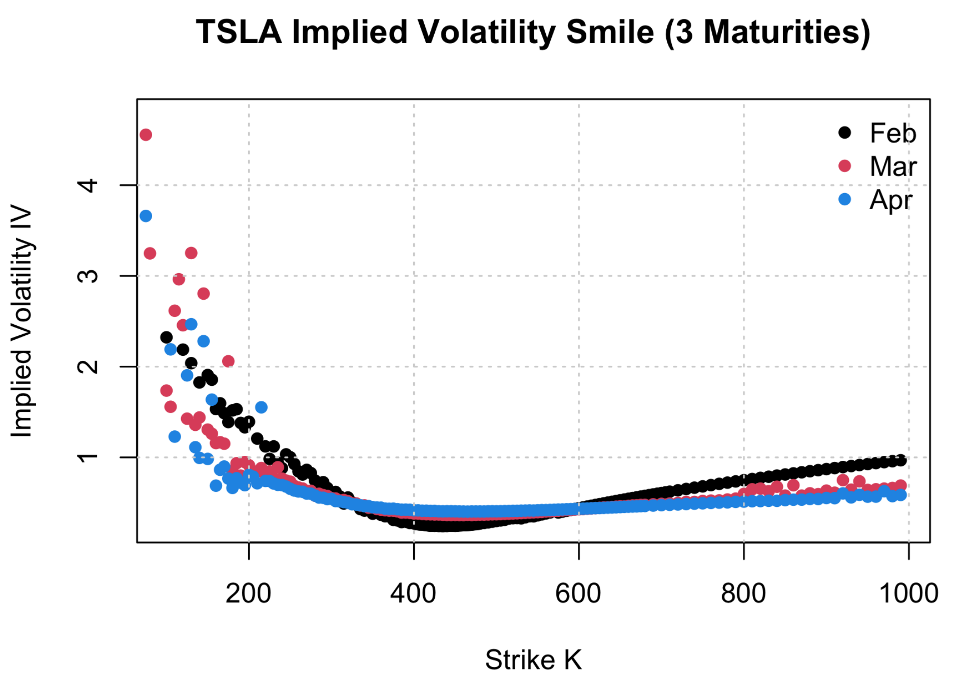

Figure 2: TSLA Implied Volatility Plot 2

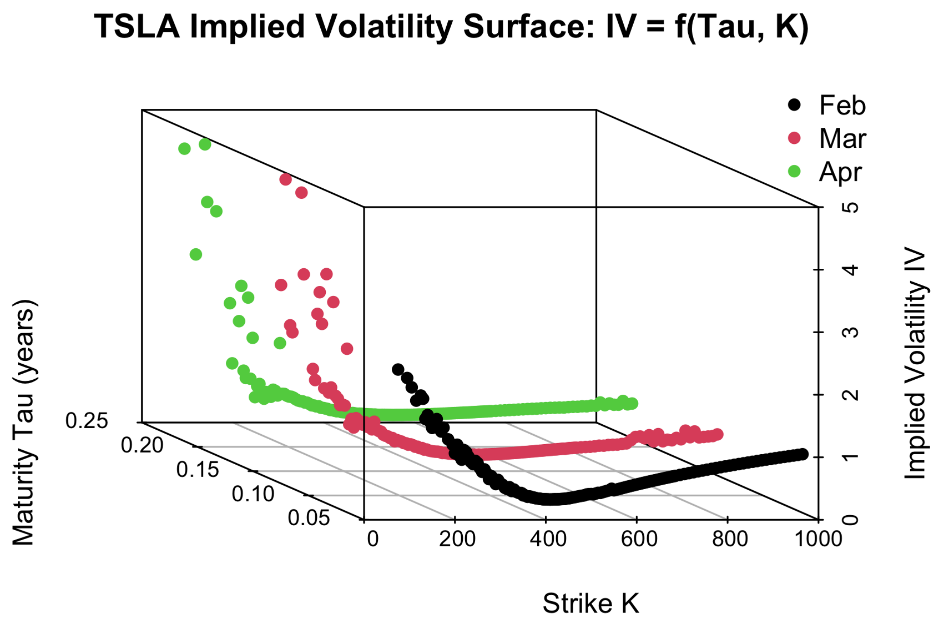

Figure 3: TSLA Implied Volatility Plot 3

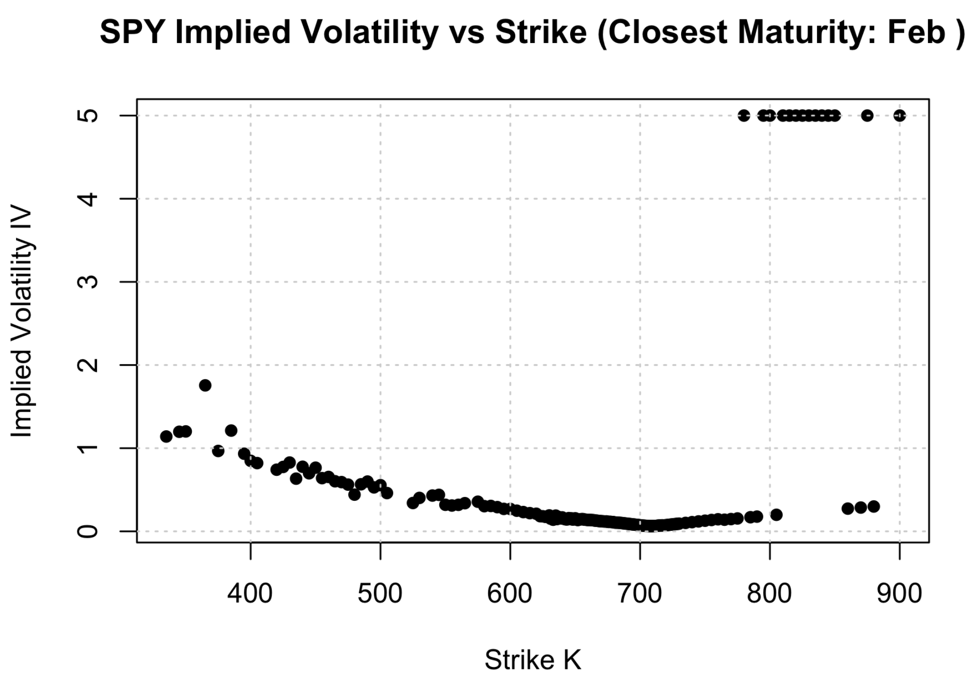

Figure 4: SPY Implied Volatility Plot 1

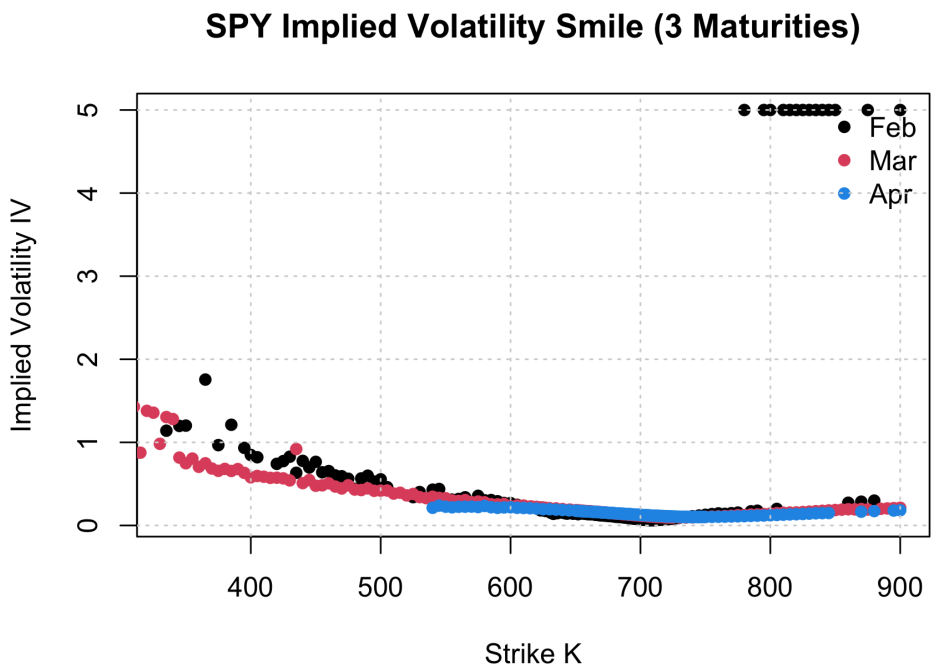

Figure 5: SPY Implied Volatility Plot 2

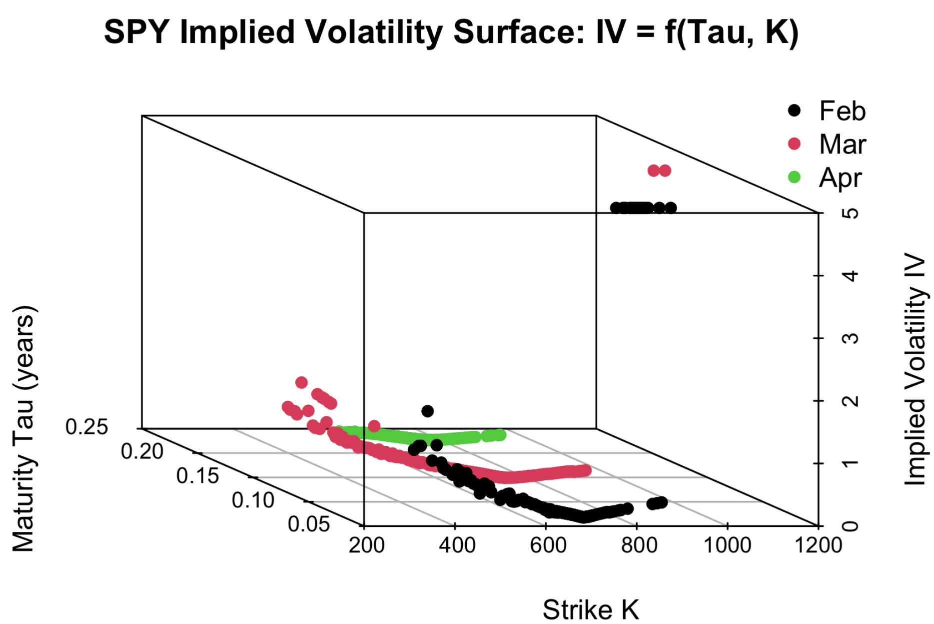

Figure 6: SPY Implied Volatility Plot 3

2.7 Question 11: Greeks

The code implementation is provided in the Appendix Section 2.

Delta represents the first order derivative of the option, gamma is the 2nd order derivative, and vega is the measure of how much an option's price changes depending on 1 percent volility change.

Most Values are the same for both methods, but there is a difference in the delta and gamma values. The Black-Scholes method seems to overpredict the values, while the Finite Difference method seems to underpredict the values. They both got the same Vega values.

Table 8: Greeks Comparison: Black–Scholes vs Finite Difference (Q11)

Symbol	Exp	Method	S_0	K	τ	σ	Δ	Γ	Vega
"SPY"	"Apr"	"BS"	684.445	684	0.213699	0.143565	0.563634	0.008671	124.617
"SPY"	"Apr"	"FD"	684.445	684	0.213699	0.143565	0.563298	0.008655	124.617
"SPY"	"Feb"	"BS"	684.445	684	0.060274	0.102693	0.549906	0.022938	66.511631
"SPY"	"Feb"	"FD"	684.445	684	0.060274	0.102693	0.548384	0.022649	66.511631
"SPY"	"Mar"	"BS"	684.445	684	0.136986	0.140157	0.553535	0.011135	100.150534
"SPY"	"Mar"	"FD"	684.445	684	0.136986	0.140157	0.553081	0.011102	100.150534
"TSLA"	"Apr"	"BS"	415.607	415	0.213699	0.412213	0.557158	0.004986	75.858709
"TSLA"	"Apr"	"FD"	415.607	415	0.213699	0.412213	0.557097	0.004985	75.858709
"TSLA"	"Feb"	"BS"	415.607	415	0.060274	0.253514	0.535797	0.015361	40.541957
"TSLA"	"Feb"	"FD"	415.607	415	0.060274	0.253514	0.535538	0.015329	40.541957
"TSLA"	"Mar"	"BS"	415.607	415	0.136986	0.373682	0.546083	0.006894	60.956629
"TSLA"	"Mar"	"FD"	415.607	415	0.136986	0.373682	0.545996	0.006891	60.956629

2.8 Question 12: Pricing Using DATA2

The code implementation is provided in the Appendix Section 2.

Table 9: Black–Scholes ATM Prices Using Estimated Volatility (Q12)

Symbol	Exp	Type	S_0	K	τ	σ	BS Price
"TSLA"	"Feb"	"Call"	430.410003662109	415	0.057534	0.253514	20.372881
"TSLA"	"Feb"	"Put"	430.410003662109	415	0.057534	0.252686	4.067319
"TSLA"	"Mar"	"Call"	430.410003662109	415	0.134247	0.373682	32.771813
"TSLA"	"Mar"	"Put"	430.410003662109	415	0.134247	0.371977	15.238689
"TSLA"	"Apr"	"Call"	430.410003662109	415	0.210959	0.412213	41.909107
"TSLA"	"Apr"	"Put"	430.410003662109	415	0.210959	0.409214	23.100466
"SPY"	"Feb"	"Call"	691.969970703125	684	0.057534	0.102693	12.469328
"SPY"	"Feb"	"Put"	691.969970703125	684	0.057534	0.100875	2.966154
"SPY"	"Mar"	"Call"	691.969970703125	684	0.134247	0.140157	20.427985
"SPY"	"Mar"	"Put"	691.969970703125	684	0.134247	0.147017	9.778697
"SPY"	"Apr"	"Call"	691.969970703125	684	0.210959	0.143565	25.390726
"SPY"	"Apr"	"Put"	691.969970703125	684	0.210959	0.155955	13.683617

3 Section 3: Numerical Integration – AMM Fee Revenue (Questions 13–15)

3.1 Question 13: Derive Swap Amounts

Provide the derivation of swap amounts x and y under Case 1 and Case 2.

Part 3(a): Liquidity changes under a fee γ

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

$$x_{t+1} = x_t - Delta_x \tag{4}$$
$$y_{t+1} = y_t + (1-\gamma)Delta_y \tag{5}$$

$$Delta_x > 0, \qquad Delta_y > 0. \tag{6}$$

Boundary Conditions

$$\frac{P_{t+1}}{1-\gamma} = \frac{y_{t+1}}{x_{t+1}} \cdot \frac{1}{1-\gamma} = S_{t+1} \tag{7}$$

Multiplying both sides by $(1-\gamma)$,

$$P_{t+1} = \frac{y_{t+1}}{x_{t+1}} = S_{t+1}(1-\gamma) \tag{8}$$

$$\frac{y_{t+1}}{x_{t+1}} = S_{t+1}(1-\gamma) \tag{9}$$

$$y_{t+1} = S_{t+1}(1-\gamma)\, x_{t+1}. \tag{10}$$

Given:

$$x_{t+1}y_{t+1} = k, \qquad y_{t+1} = \frac{k}{x_{t+1}} \tag{11}$$

Solving for x_{t+1}

$$k = S_{t+1}(1-\gamma)\, x_{t+1} \cdot x_{t+1} \tag{12}$$
$$= S_{t+1}(1-\gamma)\, x_{t+1}^2 \tag{13}$$

$$\frac{k}{S_{t+1}(1-\gamma)} = x_{t+1}^2 \tag{14}$$

$$x_{t+1} = \sqrt{\frac{k}{S_{t+1}(1-\gamma)}} \tag{15}$$

Solving for y_{t+1}

$$y_{t+1} = S_{t+1}(1-\gamma)\, \frac{k}{x_{t+1}} \tag{16}$$

$$y_{t+1}^2 = k\, S_{t+1}(1-\gamma) \tag{17}$$

$$y_{t+1} = \sqrt{k\, S_{t+1}(1-\gamma)} \tag{18}$$

Given:

$$x_{t+1} = x_t - Delta_x \tag{19}$$

$$\boxed{Delta_x(S) = x_t - \sqrt{\frac{k}{S_{t+1}(1-\gamma)}}}$$

Given:

$$y_{t+1} = y_t + (1-\gamma) Delta_y \tag{20}$$

$$\boxed{Delta_y(S) = \frac{\sqrt{kS_{t+1}(1-\gamma)} - y_t}{1-\gamma}}$$

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

$$x_{t+1} = x_t + (1-\gamma)Delta_x \tag{21}$$
$$y_{t+1} = y_t - Delta_y \tag{22}$$

$$Delta_x > 0, \qquad Delta_y > 0. \tag{23}$$

Boundary Condition

$$P_{t+1}(1-\gamma) = \frac{y_{t+1}}{x_{t+1}}(1-\gamma) = S_{t+1} \tag{24}$$

Dividing both sides by $(1-\gamma)$,

$$P_{t+1} = \frac{y_{t+1}}{x_{t+1}} = \frac{S_{t+1}}{1-\gamma} \tag{25}$$

$$\frac{y_{t+1}}{x_{t+1}} = \frac{S_{t+1}}{1-\gamma} \tag{26}$$

$$y_{t+1} = \frac{S_{t+1}}{1-\gamma}\, x_{t+1} \tag{27}$$

Given:

$$x_{t+1}y_{t+1} = k, \qquad y_{t+1} = \frac{k}{x_{t+1}} \tag{28}$$

$$k = \frac{S_{t+1}}{1-\gamma} x_{t+1}^2 \tag{29}$$

$$x_{t+1} = \sqrt{\frac{k(1-\gamma)}{S_{t+1}}} \tag{30}$$

Given:

$$x_{t+1} = x_t + (1-\gamma)Delta_x \tag{31}$$

$$x_{t+1} - x_t = (1-\gamma)Delta_x \tag{32}$$

$$Delta_x = \frac{x_{t+1} - x_t}{1-\gamma} \tag{33}$$

$$\boxed{Delta_x(S) = \frac{\sqrt{\frac{k(1-\gamma)}{S_{t+1}}} - x_t}{1-\gamma}}$$

Given:

$$y_{t+1} = y_t - Delta_y \tag{34}$$

$$y_t - y_{t+1} = Delta_y \tag{35}$$

$$\boxed{Delta_y(S) = y_t - \sqrt{\frac{kS_{t+1}}{1-\gamma}}}$$

3.2 Question 14: Expected Fee Revenue (Trapezoidal Rule)

The code implementation is provided in the Appendix Section 3.

The Expected Revenue Value is 0.0085176...

3.3 Question 15: Optimal Fee Rate

The code implementation is provided in the Appendix Section 3.

Table 10: Expected Fee Revenue $E[R]$ for Different (σ, γ)

σ	γ	$E[R]$
0.2	0.001	0.00368521805400986
0.6	0.001	0.0119233746160056
1	0.001	0.0200607211429745
0.2	0.003	0.00852203387382362
0.6	0.003	0.0329832890066009
1	0.003	0.0573837575055382
0.2	0.01	0.00943035729877614
0.6	0.01	0.0810823373662479
1	0.01	0.16068988312124

Above is a table with values requested in (c), as well as the graph below of sigma vs gamma*(sigma). The plot suggests that for low volatility, a low fee rate is optimal, but once volatility approaches approximately 0.18 the fee rate needs to increase.

Table 11: Optimal Fee Rate $\gamma^*(\sigma)$ and Maximum Expected Revenue

σ	$\gamma^*(\sigma)$	$\max E[R]$
0.2	0.01	0.00943035729877614
0.6	0.01	0.0810823373662479
1	0.01	0.16068988312124

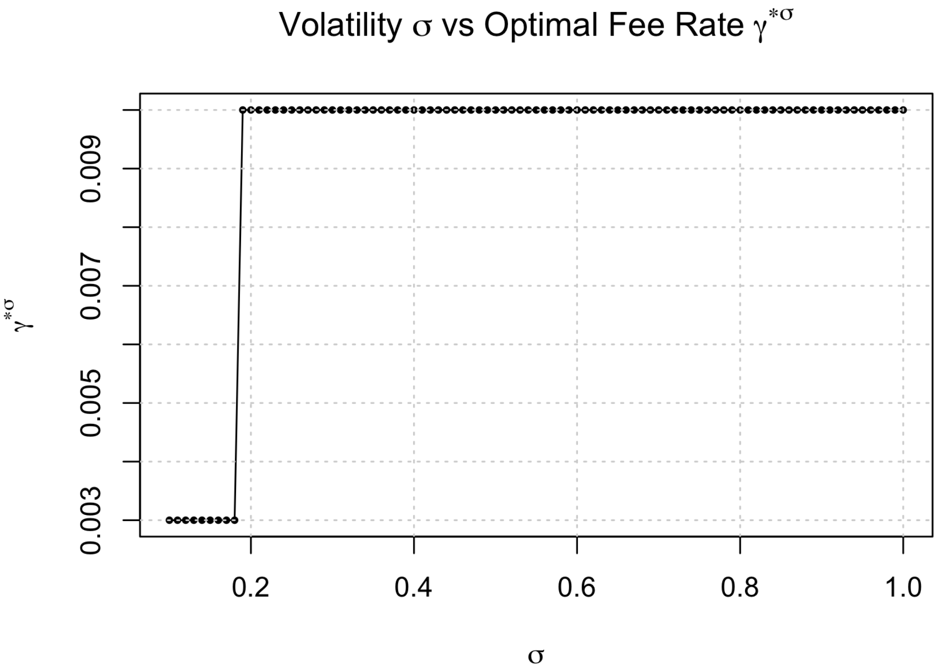

Figure 7: Optimal Fee Rate $\gamma^*(\sigma)$ as a Function of Volatility

A Appendix: R Code Implementation

This appendix contains the complete R code used to complete all parts of Homework 1.

A.1 Part 1: Data Gathering (Questions1–4)

Listing 1: Part 1: Data Gathering

```
# Part 1
############################################################
# Q1) Write a function to connect to sources and download data from one of the
# following sources: GOOGLE finance, Yahoo Finance, or Bloomberg
# I will be using Yahoo Finance

# Run the next lines if packages is are not installed
# install.packages("quantmod")
# install.packages("dplyr")
# install.packages("tidyr")

library(quantmod)
library(dplyr)
library(tidyr)

# Function for getting data from Yahoo Finance Equities
get_equity_data <- function(symbol, start_date, end_date) {
  data <- getSymbols(Symbols = symbol,
                     src = "yahoo",
                     from = start_date,
                     to = end_date,
                     auto.assign = FALSE)

  # Convert time series object into a tabular data frame with explicit date column
  data_frame <- data.frame(Date = as.Date(index(data)),
                           coredata(data),
                           row.names = NULL,
                           check.names = FALSE)

  # Clean column names to remove Symbol identifier
  names(data_frame) <- gsub("^.*\\.", "", names(data_frame))

  # Remove rows with duplicate dates - only keep first occurrence
  data_frame <- data_frame[!duplicated(data_frame$Date), ]
```

```

  # Sort by date
  data_frame <- data_frame[order(data_frame$Date), ]

  return(data_frame)
}


# Function to download multiple equity assets and combine into single frame
download_equity_data <- function(symbols, start_date, end_date) {
  sym_data <- list()  # Initialize

  # Get data for provided symbols
  for (i in seq_along(symbols)) {
    temp <- get_equity_data(symbols[i], start_date, end_date)

    # Add Ticker column
    temp$Ticker <- symbols[i]

    sym_data[[i]] <- temp
  }

  # Combine all equity data into single frame and sort by Date and Ticker
  all_data <- arrange(bind_rows(sym_data), Date, Ticker)

  return(all_data)
}

# Function for getting option chain data
get_option_data <- function(symbol, exp_date) {
  chain <- getOptionChain(Symbols = symbol, Exp = exp_date)

  # Check if option data exists
  if (is.null(chain$calls) && is.null(chain$puts)) {
    message("No options exist for ", symbol, " at expiration date", exp_date)
    return(data.frame())
  }

  # Separate data into calls and puts
  calls <- chain$calls
  puts <- chain$puts

```

```
  # Convert NULL to empty data frames for merging
  if (is.null(calls))
    calls <- data.frame()
  if (is.null(puts))
    puts <- data.frame()

  # Add label, expiration date, and ticker symbol to calls
  if (nrow(calls) > 0) {
    calls$Type <- "Call"
    calls$Expiration <- exp_date
    calls$Ticker <- symbol
  }

  # Add label, expiration date, and ticker symbol to puts
  if (nrow(puts) > 0) {
    puts$Type <- "Put"
    puts$Expiration <- exp_date
    puts$Ticker <- symbol
  }

  # Combine calls and puts into single data frame
  data_frame <- rbind(calls, puts)
  rownames(data_frame) <- NULL

  # Ensure merge produced a valid row
  if (nrow(data_frame) == 0)
    return(data.frame())

  # Clean column names to remove Symbol identifier
  names(data_frame) <- gsub("^.*\\.", "", names(data_frame))

  # Remove duplicate options
  data_frame <- data_frame[!duplicated(data_frame), ]

  # Sort data by expiration date
  data_frame <- data_frame[order(data_frame$Expiration), ]

  return(data_frame)
}


# Function to download multiple assets' option-chain data and combine
    into single frame
download_option_data <- function(symbol_expirations) {
```

```
    opt_data <- list()  # Initialize
    idx <- 1

    for (symbol in names(symbol_expirations)) {
      for (expires in symbol_expirations[[symbol]]) {
        temp <- get_option_data(symbol, expires)

        # Ensure symbol and expiration date produced data
        if (nrow(temp) > 0) {
          opt_data[[idx]] <- temp
          idx <- idx + 1  # Increment index for next data
        }
      }
    }

    # Ensure option data was collected
    if (length(opt_data) == 0)
      return(data.frame())

    # Combine option data and sort based on Ticker, Expiration, Type, Strike
    all_opts <- arrange(bind_rows(opt_data), Ticker, Expiration, Type, Strike)

    return(all_opts)
  }

# This is BONUS for Q1
# Download multiple assets, combines them with the associated time column, and
# save the data into a CSV file.

symbols <- c("TSLA", "SPY", "^VIX")

symbol_expirations <- list(
  "TSLA" = c("2026-02-20", "2026-03-20", "2026-04-17"),
  "SPY"= c("2026-02-20", "2026-03-20", "2026-04-17"),
  "^VIX" = c("2026-02-18", "2026-03-18", "2026-04-15")
  )

EQ_data <- download_equity_data(symbols, "2026-01-29", "2026-01-31")
Option_data <- download_option_data(symbol_expirations)

# Create output data folder if it does not exist
```

```
data_folder = "./data"
if (!dir.exists(data_folder))
  dir.create(data_folder)

# Save equity and option data to CSV files
write.csv(EQ_data, file.path(data_folder, "EQ_data.csv"), row.names =
    FALSE)
write.csv(Option_data, file.path(data_folder, "Option_data.csv"),
    row.names = FALSE)

##############################################################
# Q2) Using the function from part 1, pull data for TSLA, SPY, ^VIX
    splitting
# data into data1 and data2 data sets

TSLA_data1_EQ <- EQ_data[EQ_data$Ticker == "TSLA" & EQ_data$Date ==
    "2026-01-29", ]
TSLA_data2_EQ <- EQ_data[EQ_data$Ticker == "TSLA" & EQ_data$Date ==
    "2026-01-30", ]

TSLA_option_Feb <- Option_data[Option_data$Ticker == "TSLA" &
    Option_data$Expiration == "2026-02-20", ]
TSLA_option_Mar <- Option_data[Option_data$Ticker == "TSLA" &
    Option_data$Expiration == "2026-03-20", ]
TSLA_option_Apr <- Option_data[Option_data$Ticker == "TSLA" &
    Option_data$Expiration == "2026-04-17", ]

SPY_data1_EQ <- EQ_data[EQ_data$Ticker == "SPY" & EQ_data$Date ==
    "2026-01-29", ]
SPY_data2_EQ <- EQ_data[EQ_data$Ticker == "SPY" & EQ_data$Date ==
    "2026-01-30", ]

SPY_option_Feb <- Option_data[Option_data$Ticker == "SPY" &
    Option_data$Expiration == "2026-02-20", ]
SPY_option_Mar <- Option_data[Option_data$Ticker == "SPY" &
    Option_data$Expiration == "2026-03-20", ]
SPY_option_Apr <- Option_data[Option_data$Ticker == "SPY" &
    Option_data$Expiration == "2026-04-17", ]

VIX_data1_EQ <- EQ_data[EQ_data$Ticker == "^VIX" & EQ_data$Date ==
    "2026-01-29", ]
VIX_data2_EQ <- EQ_data[EQ_data$Ticker == "^VIX" & EQ_data$Date ==
    "2026-01-30", ]

```

```r
VIX_option_Feb <- Option_data[Option_data$Ticker == "^VIX" & Option_data$Expiration == "2026-02-18", ]
VIX_option_Mar <- Option_data[Option_data$Ticker == "^VIX" & Option_data$Expiration == "2026-03-18", ]
VIX_option_Apr <- Option_data[Option_data$Ticker == "^VIX" & Option_data$Expiration == "2026-04-15", ]

##############################################################
# Q4) Record underlying equity, ETF, or index price at the exact moment when data
# is downloaded and Time to Maturity

# Define function to compute underlying ETF or index price at download
get_underlying_price <- function(symbol) {
  download_price <- as.numeric(getQuote(symbol)$Last)
  return(download_price)
}

S0_TSLA <- get_underlying_price("TSLA")
S0_SPY <- get_underlying_price("SPY")
S0_VIX <- get_underlying_price("^VIX")

# Define function to compute time to maturity
time_to_maturity <- function(start_date, end_date) {
  t2m <- as.numeric(as.Date(start_date) - as.Date(end_date)) / 365
  return(t2m)
}

# Time to maturity Jan 29th
Tau_Feb1 <- time_to_maturity("2026-02-20", "2026-01-29")
Tau_Mar1 <- time_to_maturity("2026-03-20", "2026-01-29")
Tau_Apr1 <- time_to_maturity("2026-04-17", "2026-01-29")

# Time to maturity Jan 30th
Tau_Feb2 <- time_to_maturity("2026-02-20", "2026-01-30")
Tau_Mar2 <- time_to_maturity("2026-03-20", "2026-01-30")
Tau_Apr2 <- time_to_maturity("2026-04-17", "2026-01-30")

# Current Time to maturity
ttm_Feb <- time_to_maturity("2026-02-20", "2026-02-05")
ttm_Mar <- time_to_maturity("2026-03-20", "2026-02-05")
ttm_Apr <- time_to_maturity("2026-04-17", "2026-02-05")

# Save downloaded data and results for use in Part 2
```

```
save(
  TSLA_data1_EQ, TSLA_data2_EQ,
  SPY_data1_EQ, SPY_data2_EQ,
  VIX_data1_EQ, VIX_data2_EQ,

  TSLA_option_Feb, TSLA_option_Mar, TSLA_option_Apr,
  SPY_option_Feb, SPY_option_Mar, SPY_option_Apr,
  VIX_option_Feb, VIX_option_Mar, VIX_option_Apr,

  S0_TSLA, S0_SPY, S0_VIX,

  Tau_Feb1, Tau_Mar1, Tau_Apr1,
  Tau_Feb2, Tau_Mar2, Tau_Apr2,
  ttm_Feb, ttm_Mar, ttm_Apr,

  file = file.path(data_folder, "Data_for_HW.RData")
 )
```

A.2 Part 2: Option Pricing and Analysis (Questions 5–12)

Listing 2: Part 2: Option Pricing and Analysis Code

```
# Part 2
########################################################
# Q5) Black-Scholes pricing functions

bs_call <- function(S0, K, r, tau, sigma) {
  d1 <- (log(S0 / K) + (r + 0.5 * sigma^2) * tau) / (sigma * sqrt(tau))
  d2 <- d1 - sigma * sqrt(tau)
  price <- S0 * pnorm(d1) - K * exp(-r * tau) * pnorm(d2)
  return(price)
}

bs_put <- function(S0, K, r, tau, sigma) {
  d1 <- (log(S0 / K) + (r + 0.5 * sigma^2) * tau) / (sigma * sqrt(tau))
  d2 <- d1 - sigma * sqrt(tau)
  price <- K * exp(-r * tau) * pnorm(-d2) - S0 * pnorm(-d1)
  return(price)
}

########################################################
# Q6) Bisection method for arbitrary function

bisection <- function(f, lower, upper, tol = 1e-6, max_iter = 200) {
```

```
  f_low <- f(lower)
  f_up  <- f(upper)

  # If no sign change, bisection cannot guarantee a root
  if (!is.finite(f_low) || !is.finite(f_up) || (f_low * f_up > 0)) {
    return(NA_real_)
  }

  for (i in 1:max_iter) {
    mid <- (lower + upper) * 0.5
    f_mid <- f(mid)

    # Checking to see if finite
    if (!is.finite(f_mid))
      return(NA_real_)

    # Checking if tolerance value is reached
    if (abs(f_mid) < tol)
        return(mid)

    # Keep the side that has the sign change
    if (f_low * f_mid < 0) {
      upper <- mid
      f_up <- f_mid
    } else {
      lower <- mid
      f_low <- f_mid
    }
  }

  # If max iterations reached, return midpoint
  return(mid)
}


# Clean and prepare option data
prep_options <- function(option_df, S0_underlying) {

  # Removing rows with 0 or NA vol values
  op <- option_df[!is.na(option_df$Vol) & option_df$Vol > 0, ]

  # Finding Value of the Option 0.5*(Bid + Ask)
  op$Mid <- 0.5 * (op$Ask + op$Bid)

```

```r
  # Finding the distance between Strike and Spot
  op$ATM_Dist <- abs(op$Strike - S0_underlying)

  # Finding the minimum distance
  ATM_strike <- op$Strike[which.min(op$ATM_Dist)]

  # Returning the option data
  return(list(op = op, ATM_strike = ATM_strike))
}


# Getting At-The-Money Call and Put Rows
get_ATM_options <- function(op, ATM_strike) {
  ATM_call <- op[op$Strike == ATM_strike & op$Type == "Call", ]
  ATM_put  <- op[op$Strike == ATM_strike & op$Type == "Put", ]

  # Returning the call and put data
  return(list(ATM_call = ATM_call, ATM_put = ATM_put))
}


# Compute Implied Volatility via Bisection Method for single option
implied_vol_bisection <- function(market_price, Type, S0, K, r, Tau, tol = 1e-6) {

  # Define pricing error function f(sigma) = model - market
  price_err <- function(sigma) {
    if (Type == "Call")
      model_price <- bs_call(S0, K, r, Tau, sigma)
    else
      model_price <- bs_put(S0, K, r, Tau, sigma)

    return(model_price - market_price)
  }

  # Run bisection on wide range of sigma (i.e., very small vol to very large vol)
  sigma_hat <- bisection(price_err, lower = 1e-6, upper = 5, tol = tol, max_iter = 100)

  return(list(iv = sigma_hat, method = "Bisection"))
}
```

```r
# Compute Average Implied Volatility in moneyness band using Bisection
    Method
avg_iv_bisection <- function(op, S0_underlying, Tau, r, tol = 1e-6,
                             m_low = 0.95, m_high = 1.05) {
  op$Moneyness <- S0_underlying / op$Strike
  op_band <- op[op$Moneyness >= m_low & op$Moneyness <= m_high, ]

  if (nrow(op_band) == 0)
    return(NA_real_)

  op_band$IV <- NA_real_  # Initialize value
  op_band$Time <- NA_real_  # Initialize value

  for (i in 1:nrow(op_band)) {
    out_iv <- implied_vol_bisection(market_price = op_band$Mid[i],
                                    Type = op_band$Type[i],
                                    S0 = S0_underlying,
                                    K = op_band$Strike[i],
                                    r = r,
                                    Tau = Tau,
                                    tol = tol)
    op_band$IV[i] <- out_iv$iv
  }

  return(mean(op_band$IV, na.rm = TRUE))
}


# Process data for a single chain (TSLA, SPY: Feb, Mar, Apr) via
    Bisection Method
run_chain_bisection <- function(symbol, option_df, S0_underlying, Tau,
                                r, tol = 1e-6) {
  # Get prepared options data
  op_data <- prep_options(option_df, S0_underlying)
  op <- op_data$op
  ATM_strike <- op_data$ATM_strike

  # Get At-The_Money options call and put data
  ATM <- get_ATM_options(op, ATM_strike)
  ATM_call <- ATM$ATM_call
  ATM_put  <- ATM$ATM_put

  # Bisection ATM call
  IV_ATM_Call <- NA_real_  # Initialize value
```

```
  if (nrow(ATM_call) > 0) {
    out_call <- implied_vol_bisection(market_price = ATM_call$Mid,
                                      Type = "Call",
                                      S0 = S0_underlying,
                                      K = ATM_call$Strike,
                                      r = r,
                                      Tau = Tau,
                                      tol = tol)

    IV_ATM_Call <- out_call$iv
  }

  # Bisection ATM call
  IV_ATM_Put  <- NA_real_  # Initialize value
  if (nrow(ATM_put) > 0) {
    out_put <- implied_vol_bisection(market_price = ATM_put$Mid,
                                     Type = "Put",
                                     S0 = S0_underlying,
                                     K = ATM_put$Strike,
                                     r = r,
                                     Tau = Tau,
                                     tol = tol)

    IV_ATM_Put <- out_put$iv
  }

  Avg_IV_ATM <- mean(c(IV_ATM_Call, IV_ATM_Put), na.rm = TRUE)
  Avg_IV_Moneyness <- avg_iv_bisection(op, S0_underlying, Tau, r, tol,
     0.95, 1.05)

  out_results <- data.frame(Symbol = symbol,
                            S0 = S0_underlying,
                            Tau = Tau,
                            ATM_Strike = ATM_strike,
                            IV_ATM_Call = IV_ATM_Call,
                            IV_ATM_Put = IV_ATM_Put,
                            Avg_IV_ATM = Avg_IV_ATM,
                            Avg_IV_Moneyness = Avg_IV_Moneyness,
                            stringsAsFactors = FALSE)

  rownames(out_results) <- NULL
  return(out_results)
}
```

```r

# Build Bisection Method results table (TSLA + SPY, Feb/Mar/Apr)

# Load equity and option data and results generated in Part 1
load("./Data_for_HW.RData")

# Create output tables folder if it does not exist - used for report
table_folder = "./tables"

# This makes the folder for the first time you run the code
if (!dir.exists(table_folder))
  dir.create(table_folder)

# Went to website provided in Homework PDF to get short-term interest
    rate of 3.64% for Jan 29th and Jan 30th then converted interest rate
    to number (i.e., r = 3.64 / 100 = 0.0364)

r = 0.0364
tol = 1e-6  # Accuracy tolerance

# Define inputs in simple lists for chain processing
symbols  <- c("TSLA", "SPY")
expiries <- c("Feb", "Mar", "Apr")

# Option Data
option_data <- list(
  TSLA = list(Feb = TSLA_option_Feb, Mar = TSLA_option_Mar, Apr =
      TSLA_option_Apr),
  SPY  = list(Feb = SPY_option_Feb, Mar = SPY_option_Mar, Apr =
      SPY_option_Apr)
)

# Spot Price of Option Data
S0_values <- list(TSLA = S0_TSLA, SPY = S0_SPY)

# Expiration data
Tau_values <- list(Feb = Tau_Feb1, Mar = Tau_Mar1, Apr = Tau_Apr1)

# Generate results table for Bisection Method
bisection_results <- data.frame()  # Initialize value

for (sym in symbols) {
  for (exp in expiries) {
```

```
    temp <- run_chain_bisection(sym, option_data[[sym]][[exp]],
       S0_values[[sym]],
                                    Tau_values[[exp]], r, tol)
    temp$Expiries <- exp
    bisection_results <- rbind(bisection_results, temp)
  }
}

# Final output
View(bisection_results)
write.csv(bisection_results, file.path(table_folder,
   "bisection_results.csv"),
          row.names = FALSE)

########################################################
# Q7) Newton Method for arbitrary function
newton <- function(f, fprime, x0, tol = 1e-6, max_iter = 50) {
  x <- x0  # Initialize

  for (i in 1:max_iter) {
    fx <- f(x)
    fpx <- fprime(x)

    # Stop if f(x) is close to zero or derivative is very small derivative
    if (abs(fx) < tol || abs(fpx) < 1e-14) {
      return(x)
    }

    # Newton update
    x_new <- x - fx / fpx

    # Volatility must be positive
    if (!is.finite(x_new) || x_new <= 0) {
      return(NA_real_)
    }

    # Stop if change is very small (converged)
    if (abs(x_new - x) < tol) {
      return(x_new)
    }

    x <- x_new
  }

```

```
  return(x)
}


# Black-Scholes Vega (dPrice/dSigma) for call/put
bs_vega <- function(S0, K, r, tau, sigma) {
  d1 <- (log(S0 / K) + (r + 0.5 * sigma^2) * tau) / (sigma * sqrt(tau))
  return(S0 * sqrt(tau) * dnorm(d1))
}


# Compute Implied Volatility via Newton Method for ONE option
implied_vol_newton <- function(market_price, Type, S0, K, r, Tau, sigma0 = 0.20,
                               tol = 1e-6, max_iter = 50) {
  # If market price or time to maturity is non-finite or non-positive, return NA
  # since implied volatility cannot be computed
  if (!is.finite(market_price) || market_price <= 0 ||
      !is.finite(Tau) || Tau <= 0)
    return(list(iv = NA_real_, method = "Newton"))

  # Define pricing error function f(sigma) = model - market
  f <- function(sigma) {
    if (Type == "Call")
      bs_call(S0, K, r, Tau, sigma) - market_price
    else
      bs_put(S0, K, r, Tau, sigma) - market_price
  }

  # f'(sigma) = Vega(sigma)
  fprime <- function(sigma) {
    bs_vega(S0, K, r, Tau, sigma)
  }

  out <- newton(f, fprime, x0 = sigma0, tol = tol, max_iter = max_iter)
  return(list(iv = out, method = "Newton"))
}


# Compute Average Implied Volatility in moneyness band using Newton Method
avg_iv_newton <- function(op, S0_underlying, Tau, r, tol = 1e-6, sigma0 = 0.20,
                          m_low = 0.95, m_high = 1.05) {
```

```
  op$Moneyness <- S0_underlying / op$Strike
  op_band <- op[op$Moneyness >= m_low & op$Moneyness <= m_high, ]

  if (nrow(op_band) == 0)
    return(NA_real_)

  op_band$IV <- NA_real_  # Initialize value
  op_band$Time <- NA_real_  # Initialize value

  for (i in 1:nrow(op_band)) {
    out_iv <- implied_vol_newton(market_price = op_band$Mid[i],
                                 Type = op_band$Type[i],
                                 S0 = S0_underlying,
                                 K = op_band$Strike[i],
                                 r = r,
                                 Tau = Tau,
                                 sigma0 = sigma0,
                                 tol = tol)
    op_band$IV[i] <- out_iv$iv
  }

  return(mean(op_band$IV, na.rm = TRUE))
}

# Process data for a single chain (TSLA, SPY: Feb, Mar, Apr) via Newton Method
run_chain_newton <- function(symbol, option_df, S0_underlying, Tau, r,
                             tol = 1e-6, sigma0 = 0.20) {
  # Get prepared options data
  op_data <- prep_options(option_df, S0_underlying)
  op <- op_data$op
  ATM_strike <- op_data$ATM_strike

  # Get At-The_Money options call and put data
  ATM <- get_ATM_options(op, ATM_strike)
  ATM_call <- ATM$ATM_call
  ATM_put  <- ATM$ATM_put

  # Newton ATM call
  IV_ATM_Call <- NA_real_  # Initialize
  if (nrow(ATM_call) > 0) {
    out_call <- implied_vol_newton(market_price = ATM_call$Mid,
                                   Type = "Call",
                                   S0 = S0_underlying,
```

```r
                                   K = ATM_call$Strike,
                                   r = r,
                                   Tau = Tau,
                                   sigma0 = sigma0,
                                   tol = tol)

    IV_ATM_Call <- out_call$iv
  }

  # Newton ATM put
  IV_ATM_Put <- NA_real_  # Initialize
  if (nrow(ATM_put) > 0) {
    out_put <- implied_vol_newton(market_price = ATM_put$Mid,
                                  Type = "Put",
                                  S0 = S0_underlying,
                                  K = ATM_put$Strike,
                                  r = r,
                                  Tau = Tau,
                                  sigma0 = sigma0,
                                  tol = tol)

    IV_ATM_Put <- out_put$iv
  }

  Avg_IV_ATM <- mean(c(IV_ATM_Call, IV_ATM_Put), na.rm = TRUE)
  Avg_IV_Moneyness <- avg_iv_newton(op, S0_underlying, Tau, r, tol,
     sigma0,
                                    0.95, 1.05)

  out_results <- data.frame(Symbol = symbol,
                            S0 = S0_underlying,
                            Tau = Tau,
                            ATM_Strike = ATM_strike,
                            IV_ATM_Call = IV_ATM_Call,
                            IV_ATM_Put = IV_ATM_Put,
                            Avg_IV_ATM = Avg_IV_ATM,
                            Avg_IV_Moneyness = Avg_IV_Moneyness,
                            stringsAsFactors = FALSE)


  rownames(out_results) <- NULL
  return(out_results)
}

```

```
# Build Newton Method results table (TSLA + SPY, Feb/Mar/Apr)

# Initial guess for volatility [TSLA , SPY]
sigma0_vals <- c(TSLA = 0.4, SPY = 0.2)

newton_results <- data.frame()  # Initialize

for (sym in symbols) {
  # Getting correct value for sigma guess
  sigma0 <- sigma0_vals[sym]

  for (exp in expiries) {
    temp <- run_chain_newton(sym, option_data[[sym]][[exp]],
       S0_values[[sym]],
                                  Tau_values[[exp]], r, tol, sigma0)
    temp$Expiries <- exp
    newton_results <- rbind(newton_results, temp)
  }
}

# Final output
View(newton_results)
write.csv(newton_results, file.path(table_folder, "newton_results.csv"),
          row.names = FALSE)

# Run the next line if package is not installed
# install.packages("microbenchmark")
library(microbenchmark)  # Allows for microsecond times

# Compare run-time of Bisection and Newton methods averaged over N runs
compare_run_time <- function(option_df, S0_underlying, r, Tau, tol,
                                  sigma0, n_runs, table_folder){
  op_data <- prep_options(option_df, S0_underlying)
  op <- op_data$op
  ATM_strike <- op_data$ATM_strike
  ATM <- get_ATM_options(op, ATM_strike)
  ATM_call <- ATM$ATM_call

  # To obtain a statistically valid estimate of convergence time for each
     method,
  # average run-time over n_runs runs using the same data and accuracy
     tolerance
  output <- suppressWarnings(microbenchmark(
```

```
    Bisection = implied_vol_bisection(market_price = ATM_call$Mid,
                                      Type = "Call",
                                      S0 = S0_underlying,
                                      K = ATM_call$Strike,
                                      r = r,
                                      Tau = Tau,
                                      tol = tol),
    Newton = implied_vol_newton(market_price = ATM_call$Mid,
                                Type = "Call",
                                S0 = S0_underlying,
                                K = ATM_call$Strike,
                                r = r,
                                Tau = Tau,
                                sigma0 = sigma0,
                                tol = tol),
    times = n_runs,
    unit = "ms"  # milliseconds
  ))

  # Convert timing output to data frame for outputting to CSV file
  time_summary <- as.data.frame(summary(output))

  # Output to file
  colnames(time_summary) <- c("Method", "Min", "LQ", "Mean", "Median",
     "UQ", "Max", "Runs")
  View(time_summary)
  write.csv(time_summary, file.path(table_folder, "time_comparison.csv"),
     row.names = FALSE)
}

# Comparing convergence of Bisection and Newton methods
compare_run_time(TSLA_option_Feb, S0_TSLA, r, Tau_Feb1, tol, sigma0 = 0.2,
                 n = 100, table_folder = table_folder)

########################################################
# Q8) Put data in LaTex Table for comparison no need to code anything

# Generating table format
Q8_table <- bisection_results[, c("Symbol","Expiries","Tau","ATM_Strike",
                                  "IV_ATM_Call","IV_ATM_Put","Avg_IV_ATM","Avg_IV_Mone

# Sorting
Q8_table <- Q8_table[order(Q8_table$Symbol, Q8_table$Expiries), ]

```

```r
Q8_table$Tau <- round(Q8_table$Tau, 6)
Q8_table$ATM_Strike <- round(Q8_table$ATM_Strike, 2)
Q8_table$IV_ATM_Call <- round(Q8_table$IV_ATM_Call, 6)
Q8_table$IV_ATM_Put <- round(Q8_table$IV_ATM_Put, 6)
Q8_table$Avg_IV_ATM <- round(Q8_table$Avg_IV_ATM, 6)
Q8_table$Avg_IV_Moneyness <- round(Q8_table$Avg_IV_Moneyness, 6)

# Final output
rownames(Q8_table) <- NULL
View(Q8_table)
write.csv(Q8_table, file.path(table_folder, "Q8_table.csv"), row.names =
    FALSE)

# Getting the value of VIX right NOW
get_vix_now <- function() {
  if (!requireNamespace("quantmod", quietly = TRUE)) return(NA_real_)
  suppressWarnings({
    vix_xts <- quantmod::getSymbols("^VIX", src = "yahoo", auto.assign =
        FALSE)
    as.numeric(tail(quantmod::Cl(vix_xts), 1))
  })
}

VIX_now <- get_vix_now()
message(sprintf("Current ^VIX (Yahoo close): %.2f", VIX_now))

#########################################################
# Q9) Put-Call Parity

# Put-Call Parity (no dividends): C - P = S0 - K*exp(-r*Tau)
pc_parity_12rows <- function(symbol, option_df, S0_underlying, Tau, r) {
  # Get prepared options data
  op_data <- prep_options(option_df, S0_underlying)
  op <- op_data$op
  K <- op_data$ATM_strike

  # Get At-The_Money options call and put data
  ATM <- get_ATM_options(op, K)
  call <- ATM$ATM_call
  put <- ATM$ATM_put

  PVK <- K * exp(-r * Tau)

  # Parity prices
```

```
  P_price <- call$Mid - S0_underlying + PVK
  C_price <- put$Mid + S0_underlying - PVK

  # Call row
  call_row <- data.frame(Symbol = symbol,
                         Type = "Call",
                         Price = C_price,
                         stringsAsFactors = FALSE)

  # Put row
  put_row <- data.frame(Symbol = symbol,
                        Type = "Put",
                        Price = P_price,
                        stringsAsFactors = FALSE)

  rbind(call_row, put_row)
}

# Build final Q9 table (12 rows, minimal)
Q9_display <- data.frame()

for (sym in symbols) {
  for (exp in expiries) {
    temp <- pc_parity_12rows(sym, option_data[[sym]][[exp]],
        S0_values[[sym]],
                                 Tau_values[[exp]], r)
    temp$Month <- exp
    Q9_display <- rbind(Q9_display, temp)
  }
}

# Round prices to 4 digits after decimal
Q9_display$Price <- round(Q9_display$Price, 4)

# Final output
rownames(Q9_display) <- NULL
View(Q9_display)
write.csv(Q9_display, file.path(table_folder, "Q9_display.csv"),
          row.names = FALSE)

#########################################################
# Q10) Implied volatility plots vs strike K
#     1) 2D plot of IV vs K for the closest maturity (Feb)
#     2) 2D plot of IV vs K for Feb/Mar/Apr on the same plot (3 colors)
```

```r
#     3) 3D plot of IV as a function of both K and Tau

# Create output plot folder if it does not exist - used for report
plot_folder = "./figures"
if (!dir.exists(plot_folder))
  dir.create(plot_folder)

Type_plot <- "Call"  # Keep consistent with Q6/Q7 focus on calls
closest_exp <- "Feb"  # Closest-to-maturity in my setup


# Computing implied volatility for ALL strikes in one chain (one maturity)
compute_iv_all_strikes <- function(op, S0_underlying, Tau, r, Type = "Call",
                                   tol = 1e-6) {
  strikes <- sort(unique(op$Strike))

  out <- data.frame(Strike = strikes,
                    IV = NA_real_,
                    stringsAsFactors = FALSE)

  for (j in 1:length(strikes)) {
    K_j <- strikes[j]
    row_j <- op[op$Strike == K_j & op$Type == Type, ]
    if (nrow(row_j) == 0) next

    out$IV[j] <- implied_vol_bisection(market_price = row_j$Mid,
                                       Type = Type,
                                       S0 = S0_underlying,
                                       K = K_j,
                                       r = r,
                                       Tau = Tau,
                                       tol = tol)$iv
  }

  return(out)
}


# Building implied vol data for Feb/Mar/Apr (for one symbol)
build_iv_data_Q10 <- function(symbol, option_data_sym, S0_underlying, Tau_values,
                              r, tol = 1e-6, Type = "Call") {
  iv_all <- data.frame()  # Initialize
```

```r

  for (exp in names(option_data_sym)) {
    op_temp <- prep_options(option_data_sym[[exp]], S0_underlying)$op

    iv_df <- compute_iv_all_strikes(op = op_temp,
                                     S0_underlying = S0_underlying,
                                     Tau = Tau_values[[exp]],
                                     r = r,
                                     Type = Type,
                                     tol = tol)

    iv_df$Symbol <- symbol
    iv_df$Expiries <- exp
    iv_df$Tau <- Tau_values[[exp]]
    iv_all <- rbind(iv_all, iv_df)
  }

  rownames(iv_all) <- NULL
  return(iv_all)
}

# Building implied vol data (TSLA + SPY)
Q10_TSLA <- build_iv_data_Q10(symbol = "TSLA",
                              option_data_sym = option_data$TSLA,
                              S0_underlying = S0_values$TSLA,
                              Tau_values = Tau_values,
                              r = r,
                              tol = tol,
                              Type = Type_plot)

Q10_SPY <- build_iv_data_Q10(symbol = "SPY",
                             option_data_sym = option_data$SPY,
                             S0_underlying = S0_values$SPY,
                             Tau_values = Tau_values,
                             r = r,
                             tol = tol,
                             Type = Type_plot)


# Plot 1: Closest maturity only (IV vs K) for closest maturity
plot_iv_closest_maturity <- function(iv_data, symbol, exp_to_plot,
    plot_folder) {
  df <- iv_data[iv_data$Symbol == symbol & iv_data$Expiries ==
      exp_to_plot, ]
```

```
  # Keep only good points (Strike, IV must all be finite)
  df <- df[is.finite(df$Strike) & is.finite(df$IV), ]

  # Setup PNG output file
  plot_filename <- paste0("plot_1_", symbol, ".png")
  full_path <- file.path(plot_folder, plot_filename)
  png(full_path, width = 6, height = 4.5, units = "in", res = 300)

  # Generate plot
  plot(df$Strike, df$IV, xlab = "Strike K", ylab = "Implied Volatility IV",
       main = paste(symbol, "Implied Volatility vs Strike (Closest Maturity:", exp_to_plot, ")"),
       pch = 16)
  grid()
  invisible(dev.off())  # Close output file
}

# Plot data
plot_iv_closest_maturity(Q10_TSLA, "TSLA", closest_exp, plot_folder)
plot_iv_closest_maturity(Q10_SPY,  "SPY",  closest_exp, plot_folder)


# Plot 2: Feb/Mar/Apr on same plot (3 colors, 3 sets of points)
plot_iv_three_maturities <- function(iv_data, symbol, plot_folder) {
  df <- iv_data[iv_data$Symbol == symbol, ]

  # Keep only good points (Strike, IV, Tau must all be finite)
  df <- df[is.finite(df$Strike) & is.finite(df$IV) & is.finite(df$Tau), ]

  # Separate individual maturities
  df_feb <- df[df$Expiries == "Feb" & is.finite(df$IV), ]
  df_mar <- df[df$Expiries == "Mar" & is.finite(df$IV), ]
  df_apr <- df[df$Expiries == "Apr" & is.finite(df$IV), ]

  y_min <- min(df$IV, na.rm = TRUE)
  y_max <- max(df$IV, na.rm = TRUE)

  # Setup PNG output file
  plot_filename <- paste0("plot_2_", symbol, ".png")
  full_path <- file.path(plot_folder, plot_filename)
  png(full_path, width = 6, height = 4.5, units = "in", res = 300)

```

```
  # Generate plot
  plot(df_feb$Strike, df_feb$IV, xlab = "Strike K", ylab = "Implied Volatility IV",
       main = paste(symbol, "Implied Volatility Smile (3 Maturities)"),
       pch = 16, ylim = c(y_min, y_max))

  points(df_mar$Strike, df_mar$IV, col = 2, pch = 16)
  points(df_apr$Strike, df_apr$IV, col = 4, pch = 16)

  legend("topright", legend = c("Feb", "Mar", "Apr"),
         col = c(1, 2, 4), pch = 16, bty = "n")
  grid()
  invisible(dev.off())  # Close output file
}

# Plot data
plot_iv_three_maturities(Q10_TSLA, "TSLA", plot_folder)
plot_iv_three_maturities(Q10_SPY,  "SPY", plot_folder)

# BONUS 3D Plot
# 3D plot library package
# Run the next line if package is not installed
# install.packages("scatterplot3d")

library(scatterplot3d)

# Plot 3: 3D plot of Implied Volatility as a function of both Strike and Maturity
plot_iv_3d <- function(iv_data, symbol, expiries, plot_folder) {
  df <- iv_data[iv_data$Symbol == symbol, ]

  # Keep only good points (Strike, IV, Tau must all be finite)
  df <- df[is.finite(df$Strike) & is.finite(df$IV) & is.finite(df$Tau), ]

  # Ensure consistent color mapping for maturities
  df$Expiries <- factor(df$Expiries, levels = expiries)
  cols <- as.numeric(df$Expiries)  # 1 - Feb, 2 - Mar, 3 - Apr

  # Setup PNG output file
  plot_filename <- paste0("plot_3_", symbol, ".png")
  full_path <- file.path(plot_folder, plot_filename)
  png(full_path, width = 6, height = 4.5, units = "in", res = 300)

  # 3D scatter: x = K, y = Tau, z = IV
```

```r
  par(mar = c(5, 8, 4, 8))  # Increase right margin for legend
  scatterplot3d::scatterplot3d(df$Strike, df$Tau, df$IV,
    xlab = "Strike K", ylab = "Maturity Tau (years)", zlab = "Implied
       Volatility IV",
    main = paste(symbol, "Implied Volatility Surface: IV = f(Tau, K)"),
    pch = 16, color = cols, angle = 145)

  par(xpd = NA)
  legend("topright", inset = c(-0.04, -0.05),
         legend = expiries, col = seq_along(expiries),
         pch = 16, bty = "n")

  invisible(dev.off())  # Close output file
}

# Plotting data
plot_iv_3d(Q10_TSLA, "TSLA", expiries, plot_folder)
plot_iv_3d(Q10_SPY, "SPY", expiries, plot_folder)

########################################################
# Q11) Greeks (CALL): Delta, Gamma, Vega
# Compare Black-Scholes closed-form (BS) vs Finite Difference (FD)

bs_call_greeks <- function(S0, K, r, tau, sigma) {
  d1 <- (log(S0 / K) + (r + 0.5 * sigma^2) * tau) / (sigma * sqrt(tau))

  # Sensitivity of option price to underlying price
  Delta <- pnorm(d1)

  # Curvature of option price w.r.t. underlying price
  Gamma <- dnorm(d1) / (S0 * sigma * sqrt(tau))

  # Sensitivity of option price to volatility
  Vega <- S0 * dnorm(d1) * sqrt(tau)

  return(list(Delta = Delta, Gamma = Gamma, Vega = Vega))
}


fd_call_greeks <- function(S0, K, r, tau, sigma, hS_frac = 0.01, hSigma =
    1e-4) {
  hS <- hS_frac * S0

  C0 <- bs_call(S0, K, r, tau, sigma)
```

```
  C_up <- bs_call(S0 + hS, K, r, tau, sigma)
  C_dn <- bs_call(S0 - hS, K, r, tau, sigma)

  # Sensitivity of option price to underlying price
  Delta <- (C_up - C_dn) / (2 * hS)

  # Curvature of option price with respect to underlying price
  Gamma <- (C_up - 2*C0 + C_dn) / (hS^2)

  C_sig_up <- bs_call(S0, K, r, tau, sigma + hSigma)
  C_sig_dn <- bs_call(S0, K, r, tau, sigma - hSigma)

  # Sensitivity of option price to volatility
  Vega <- (C_sig_up - C_sig_dn) / (2 * hSigma)

  return(list(Delta = Delta, Gamma = Gamma, Vega = Vega))
}

#  Creating columns for the table
Q11_table <- data.frame()  # Initialize

for (sym in symbols) {
  for (exp in expiries) {
    row_res <- bisection_results[bisection_results$Symbol == sym &
                                  bisection_results$Expiries == exp, ]
    S0_use <- row_res$S0
    K_use <- row_res$ATM_Strike
    Tau_use <- row_res$Tau
    sig_use <- row_res$IV_ATM_Call

    g_bs <- bs_call_greeks(S0_use, K_use, r, Tau_use, sig_use)
    g_fd <- fd_call_greeks(S0_use, K_use, r, Tau_use, sig_use)

    row_bs <- data.frame(Symbol = sym,
                         Expiries = exp,
                         Greek_Method = "BS",
                         S0 = S0_use,
                         K = K_use,
                         Tau = Tau_use,
                         Sigma = sig_use,
                         Delta = g_bs$Delta,
                         Gamma = g_bs$Gamma,
                         Vega  = g_bs$Vega,
                         stringsAsFactors = FALSE)
```

```
    row_fd <- data.frame(Symbol = sym,
                         Expiries = exp,
                         Greek_Method = "FD",
                         S0 = S0_use,
                         K = K_use,
                         Tau = Tau_use,
                         Sigma = sig_use,
                         Delta = g_fd$Delta,
                         Gamma = g_fd$Gamma,
                         Vega  = g_fd$Vega,
                         stringsAsFactors = FALSE)

    Q11_table <- rbind(Q11_table, row_bs, row_fd)
  }
}

Q11_table$S0 <- round(Q11_table$S0, 3)
Q11_table$K <- round(Q11_table$K, 3)
Q11_table$Tau <- round(Q11_table$Tau, 6)
Q11_table$Sigma <- round(Q11_table$Sigma, 6)
Q11_table$Delta <- round(Q11_table$Delta, 6)
Q11_table$Gamma <- round(Q11_table$Gamma, 6)
Q11_table$Vega <- round(Q11_table$Vega, 6)

Q11_table <- Q11_table[order(Q11_table$Symbol, Q11_table$Expiries,
    Q11_table$Greek_Method), ]

# Final output
rownames(Q11_table) <- NULL
View(Q11_table)
write.csv(Q11_table, file.path(table_folder, "Q11_table.csv"),
          row.names = FALSE)

########################################################
# Q12) Process second day of equity and option data

# DATA2 short-term interest rate (Jan 30)
r <- 0.0364

# DATA2 spot prices from your saved equity data
S0_TSLA_2 <- as.numeric(tail(TSLA_data2_EQ$Close, 1))
S0_SPY_2 <- as.numeric(tail(SPY_data2_EQ$Close, 1))

```

```r
S0_values2 <- list(TSLA = S0_TSLA_2, SPY = S0_SPY_2)
Tau_values2 <- list(Feb = Tau_Feb2, Mar = Tau_Mar2, Apr = Tau_Apr2)

# Q12: price DATA2 ATM options using DATA1 implied vols (by maturity)
Q12_ATM_prices <- data.frame()  # Initialize

for (sym in symbols) {
  for (exp in expiries) {
    row1 <- bisection_results[bisection_results$Symbol == sym &
                                bisection_results$Expiries == exp, ]
    K_ATM <- row1$ATM_Strike

    sigma_call <- row1$IV_ATM_Call   # DATA1 sigma
    sigma_put  <- row1$IV_ATM_Put    # DATA1 sigma

    S0_2  <- S0_values2[[sym]]          # DATA2 spot
    Tau_2 <- Tau_values2[[exp]]         # DATA2 tau

    # Price in DATA2 using DATA1 sigma
    C2 <- bs_call(S0 = S0_2, sigma = sigma_call, tau = Tau_2, K = K_ATM, r = r)
    P2 <- bs_put(S0 = S0_2, sigma = sigma_put, tau = Tau_2, K = K_ATM, r = r)

    Q12_ATM_prices <- rbind(Q12_ATM_prices,
                            data.frame(Symbol=sym, Expiries=exp, Type="Call",
                                       S0_DATA2=S0_2, K=K_ATM, Tau_DATA2=Tau_2, Sigma_DATA1=sigma_call,
                                       BS_Price_DATA2=C2, stringsAsFactors=FALSE),
                            data.frame(Symbol=sym, Expiries=exp, Type="Put",
                                       S0_DATA2=S0_2, K=K_ATM, Tau_DATA2=Tau_2, Sigma_DATA1=sigma_put,
                                       BS_Price_DATA2=P2, stringsAsFactors=FALSE))
  }
}

Q12_ATM_prices$BS_Price_DATA2 <- round(Q12_ATM_prices$BS_Price_DATA2, 6)
Q12_ATM_prices$Sigma_DATA1 <- round(Q12_ATM_prices$Sigma_DATA1, 6)
```

```r
Q12_ATM_prices$Tau_DATA2 <- round(Q12_ATM_prices$Tau_DATA2, 6)

# Final output
rownames(Q12_ATM_prices) <- NULL
View(Q12_ATM_prices)
write.csv(Q12_ATM_prices, file.path(table_folder, "Q12_ATM_prices.csv"),
    row.names = FALSE)
```

A.3 Part 3: Numerical Integration and AMM Fee Analysis (Questions 13–15)

Listing 3: Part 3: AMM Fee Revenue (Trapezoidal Rule) and Optimal Fee Rate

```r

# Part 3
########################################################
# (b) Expected fee revenue (minimal version)

# Given
x0 <- 1000
y0 <- 1000
St <- 1
dt <- 1/365

sigma <- 0.2
gamma <- 0.003

# Trapezoid rule
trapz <- function(a, b, n, f) {
  h <- (b - a) / n
  s <- a + h * (0:n)
  vals <- f(s)
  h * (0.5 * vals[1] + sum(vals[2:n]) + 0.5 * vals[n + 1])
}

# CASE 1: s > P / (1-gamma)
case1 <- function(s, x, y, gamma) {
  k <- x * y
  dx <- x - sqrt(k / ((1 - gamma) * s))
  dy <- (sqrt(k * (1 - gamma) * s) - y) / (1 - gamma)
  list(dx = dx, dy = dy)
}

# CASE 2: s < P * (1-gamma)
case2 <- function(s, x, y, gamma) {
```

```r
  k <- x * y
  dx <- (sqrt(k * (1 - gamma) / s) - x) / (1 - gamma)
  dy <- y - sqrt(k * s / (1 - gamma))
  list(dx = dx, dy = dy)
}

# Lognormal density for S_{t+1} given S_t = St
log_density <- function(s, sigma, dt) {
  mu  <- log(St) - 0.5 * sigma^2 * dt
  var <- sigma^2 * dt
  (1 / (s * sqrt(2 * pi * var))) * exp(-(log(s) - mu)^2 / (2 * var))
}

# Piecewise revenue function R(s)
R_func <- function(s, x, y, gamma) {
  P <- y / x
  upper <- P / (1 - gamma)
  lower <- P * (1 - gamma)

  R_values <- numeric(length(s))

  for (i in seq_along(s)) {
    if (s[i] > upper) {
      result <- case1(s[i], x, y, gamma)
      R_values[i] <- gamma * result$dy
    } else if (s[i] < lower) {
      result <- case2(s[i], x, y, gamma)
      R_values[i] <- gamma * result$dx * s[i]
    } else {
      R_values[i] <- 0
    }
  }

  R_values
}

# Expected Revenue E[R(S_{t+1})]
expected_revenue <- function(x, y, sigma, gamma, dt, a = 0.1, b = 2.0, n = 20000) {
  integrand <- function(s) {
    R_func(s, x, y, gamma) * log_density(s, sigma, dt)
  }
  trapz(a, b, n, integrand)
}
```

```r

#Computing ER for one (sigma, gamma)
ER <- expected_revenue(x = x0, y = y0, sigma = sigma, gamma = gamma, dt = dt,
                       a = 0.1, b = 2.0, n = 20000)
message("Expected Revenue (Part b): ", ER)

########################################################
# Part 3 (c): Optimal Fee R\te under different volatilities

# Given sets in the problem
sigmas <- c(0.2, 0.6, 1.0)
gammas <- c(0.001, 0.003, 0.01)

# Generate Grid
grid <- expand.grid(sigma = sigmas, gamma = gammas)
grid$ER <- numeric(nrow(grid))

# Populate expected revenue values
for (row in 1:nrow(grid)) {
  grid$ER[row] <- expected_revenue(x = x0, y = y0,
                                   sigma = grid$sigma[row],
                                   gamma = grid$gamma[row],
                                   dt = dt,
                                   a = 0.1, b = 2.0, n = 20000)
}

# Best gamma*(sigma) among the discrete gamma options
best <- data.frame(sigma = numeric(),
                   gamma_star = numeric(),
                   ER_max = numeric())

for (s in sigmas) {
  sub <- grid[grid$sigma == s, ]
  idx <- which.max(sub$ER)

  best <- rbind(best, data.frame(sigma = s,
                                 gamma_star = sub$gamma[idx],
                                 ER_max = sub$ER[idx]))
}

cat("\nPart (c) Table of E[R] values:\n")
print(grid)

```

```r
cat("\nPart (c) Best gamma*(sigma) among the 3 options:\n")
print(best)

# Saving tables
table_folder <- "./tables"
if (!dir.exists(table_folder)) dir.create(table_folder, recursive = TRUE)

write.csv(grid, file.path(table_folder, "ER_grid_table.csv"), row.names = FALSE)
write.csv(best, file.path(table_folder, "best_gamma_table.csv"), row.names = FALSE)

# Plotting gamma*(sigma) over a sigma grid using the same discrete gamma set

sigma_grid <- seq(0.1, 1.0, by = 0.01)
gamma_star_grid <- numeric(length(sigma_grid))

for (ii in seq_along(sigma_grid)) {
  ERs <- numeric(length(gammas))

  for (jj in seq_along(gammas)) {
    ERs[jj] <- expected_revenue(x = x0, y = y0,
                                sigma = sigma_grid[ii],
                                gamma = gammas[jj],
                                dt = dt,
                                a = 0.1, b = 2.0, n = 20000)
  }

  gamma_star_grid[ii] <- gammas[which.max(ERs)]
}

out <- data.frame(sigma = sigma_grid, gamma_star = gamma_star_grid)

# Saving plot
plot_folder <- "./figures"
if (!dir.exists(plot_folder)) dir.create(plot_folder, recursive = TRUE)

full_path <- file.path(plot_folder, "optimal_fee.png")
png(full_path, width = 6, height = 4.5, units = "in", res = 300)

plot(out$sigma, out$gamma_star,
     xlab = expression(sigma),
     ylab = expression(gamma^"*"(sigma)),
```

```
    main = expression("Volatility " * sigma * " vs Optimal Fee Rate " *
        gamma^"*"(sigma)),
    type = "o", pch = 16, cex = 0.6)

grid()
invisible(dev.off())
```