Homework Assignment 1
FE 621: Computational Methods in Finance
Instructor: Ionut Florescu
2/20/2019

Rukmal Weerawarana
rweerawa@stevens.edu | 104-307-27
Department of Financial Engineering
Stevens Institute of Technology

Overview

In this Homework Assignment, we explore various numerical optimization methods through the lens of the Black-Scholes-Merton Option pricing model (Shreve 2004). Using this, we calculate explore the implied volatility of options for various assets traded on the market. Furthermore, we also explore numeric methods of differential calculation to compute the Greeks of these candidate options. Finally, we explore numeric integration and the behavior of various quadrature methods.

Unless otherwise stated, the following shorthand notation is used to distinguish between dates:

- **DATA1** - Wednesday, February 6 2019 (*2/6/19*)
- **DATA2** - Thursday, February 7 2019 (*2/7/19*)

The content of this Homework Assignment is divided into three sections; the first discusses data gathering, formatting, and a discussion of the assets being examined. The second contains data analysis, and an exploration of implied volatility through the Black-Scholes-Merton pricing framework and related computations. Finally, the third section discusses numerical integration and the convergence of various quadrature rules.

See Appendix A for specific question implementations, and (Weerawarana 2019) for full source code of the fe621 *Python package.*

1 Data Overview

1.1 Asset Descriptions

1.1.1 *SPY* - SPDR S&P 500 ETF (State Street Global Advisors 2019)

The S&P 500 (i.e. *Standard & Poor's 500*) is a stock market index tracking the 500 largest companies on the American Stock Exchange by Market Capitalization. In this case, the market capitalization is defined as the number of outstanding shares, multiplied by the current share price. A stock market index is designed to be a metric that can be used by market observers as a benchmark to gauge the relative health of the stock market, by analyzing the aggregate performance of its largest components.

However, this index is not the same as the *SPY* ETF. An ETF (*Exchange Traded Fund*) is a basket of stocks that is designed to track a specific index or benchmark. That is, it provides investors with exposure to a index or benchmark, without having to own all of the underlying assets that constitute a composite ETF. In addition to higher liquidity, this type of investment also provides lower transaction costs and required minimum investment to gain exposure to a given index or benchmark. It is traded on an exchange, akin to a typical traded asset.

1.1.2 *VIX* - CBOE Volatility Index (CBOE (Chicago Board Options Exchange) 2019)

The CBOE (*Chicago Board Options Exchange*) volatility index, *VIX* is an exchange traded product (*ETP*) designed to give investors exposure to the market's expectation of 30-day volatility. It is priced using a large set of implied volatility of put and call options on the S&P 500 index to gauge investor sentiment. Typically, the price of the VIX has an inverse relationship to the price of the S&P 500 index. Similar to an ETF, an ETP is also traded on an exchange as a typical traded asset.

1.2 Data Gathering

For the assignment, we downloaded monthly options on *Amazon Inc.* (ticker: AMZN) and *S&P 500 ETF* (ticker: SPY) at various strike prices for the following dates:

- *02/15/19* - Friday, February 15 2019;
- *03/15/19* - Friday, March 15 2019;
- *04/18/19* - Thursday, April 18 2019.

A wide variety of option strike prices were considered, with the following ranges:

- *AMZN* - \$1555 to \$1725 in increments of \$5 (35 strike prices);
- *SPY* - \$256 to \$284 in increments of \$1 (29 strike prices).

Intra-day minute closing price data was gathered for both put and call options with expiration dates and strike prices detailed above. This intra-day data was gathered for the trading day *2/6/19* (February 6 2019; **DATA1**). Additionally, intra-day minute closing price data was also downloaded for each of the underlying assets. This data was downloaded for both *2/6/19* (February 6 2019; **DATA1**), and *2/7/19* (February 7 2019; **DATA2**).

This data detailed above was gathered utilizing *Rblpapi* (Armstrong et al. 2018), which provides an R interface to data on the Bloomberg Terminal (Bloomberg L.P. 2019). The data download was automated, and corresponding intra-day prices for each of the options were output to individual files. The source code for this implementation is available in Appendix A.1.1.

Furthermore, as a proxy for the *risk-free rate*, we chose to utilize the effective Federal Funds Rate (FFR). This is the interest rate at which depository institutions in the United States lend reserve balances to other depository institutions overnight. This data was gathered for both dates, and correspond to **DATA1** and **DATA2**. The effective FFR is published daily by the US Federal Reserve Board of Governors, and are expressed as yields per annum (Board of Governors of the Federal Reserve System 2019).

1.2.1 Data Cleaning

For easier programmatic access, the data was placed in a hierarchical structure, corresponding to the **DATA1**, **DATA2** data division. Each of the option and asset prices for the corresponding days were placed in the requisite sub-folders. This directory structure is reproduced below.

```
data
├── DATA1
│   ├── AMZN
│   ├── SPY
│   └── VIX
└── DATA2
    ├── AMZN
    ├── SPY
    └── VIX

8 directories
```

Option price filenames were changed to OOC format option names, discussed further below. This was done utilizing a cleaning script, written in Python. This script employs utility functions from the *fe621* Python package (Weerawarana 2019). The cleaning script is reproduced in Appendix A.1.2.

1.3 Option Naming Convention

A modern convention for naming option contracts was proposed by the Options Clearing Commission (OCC) in 2008 (Options Symbology Initiative Working Group 2008), and adopted in 2010. The OCC is an organization that acts as both the issuer and guarantor for option and future contracts. The OCC is governed by the Securities and Exchange Commission (SEC) and the Commodities Futures Trading Commission (CFTC). The current convention for option naming is best explained by example.

Consider the option code, *AMZN190215C01960000*. This corresponds to a **Call Option** on **Amazon Inc. (AMZN)**, with a strike price of **$1960.00** and an expiration date of **2/15/19** (February 15 2019).

The methodology of this nomenclature is explained in detail below:

AMZN190215C01960000

- **AMZN** - Ticker of the company (arbitrary length; always first sequence of characters)
- **19** - Expiration year of the contract (shortened to two digits, i.e. 2019 → 19)
- **02** - Expiration month of the contract
- **15** - Expiration day of the contract
- **C** - Type of option (*C* for call, *P* for put)
- **01960** - Dollar component of strike price (in $; always 5 digits)

- **000** - $\frac{1}{1000}^{\text{th}}$ Dollar component of strike price (in $\frac{1}{1000}$\$; always 3 digits)

Similarly, the following option code corresponds to a **Put Option** on **SPDR S&P 500 ETF (SPY)**, with a strike price of **$287.50** and an expiration date of **3/15/19** (March 15 2019):

SPY190315P00287500

Finally, the following option code corresponds to a **Call Option** on **CBOE Volatility Index (VIX)**, with a strike price of **$16.35** and an expiration date of **4/18/19** (February 18 2019):

VIX190418C00016350

2 Data Analysis

Note: All Python scripts reproduced in this section are extracted from the fe621 (Weerawarana 2019) *package created for this class.*

2.1 Black-Scholes Model Formulas

With the probabilities d_1 and d_2 defined as:

$$d_1 = \frac{\log\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

$$\Phi(x) = \int_{-\infty}^{x} \phi(z)dz = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{\frac{-z^2}{2}} dz$$

```python
from typing import Tuple

import numpy as np


def computeD1D2(current: float, volatility: float, ttm: float, strike: float,
                rf: float) -> Tuple[float, float]:
    """Helper function to compute the risk-adjusted priors of exercising the
    option contract, and keeping the underlying asset. This is used in the
    computation of both the Call and Put options in the
    Black-Scholes-Merton framework.

    Arguments:
        current {float} -- Current price of the underlying asset.
        volatility {float} -- Volatility of the underlying asset price.
        ttm {float} -- Time to expiration (in years).
        strike {float} -- Strike price of the option contract.
        rf {float} -- Risk-free rate (annual).

    Returns:
        Tuple[float, float] -- Tuple with d1, and d2 respectively.
    """

    d1 = (np.log(current / strike) + (rf + ((volatility ** 2) / 2)) * ttm) \
        / (volatility * np.sqrt(ttm))
    d2 = d1 - (volatility * np.sqrt(ttm))

    return (d1, d2)
```

../fe621/black_scholes/util.py

Note: **The following assumes the dividend rate, $q = 0$.**

2.1.1 Put Option

The Black-Scholes Option price for a European Put ($P(S_t)$) option is defined as:

$$P(S_t) = Ke^{-r(T-t)}\Phi(-d_2) - S_t\Phi(-d_1)$$

```python
from .util import computeD1D2

from scipy.stats import norm
import numpy as np


def blackScholesPut(current: float, volatility: float, ttm: float,
                    strike: float, rf: float) -> float:
    """Function to compute the Black-Scholes-Merton price of a European Put
    Option, parameterized by the current underlying asset price, volatility,
    time to expiration, strike price, and risk-free rate.

    Arguments:
        current {float} -- Current price of the underlying asset.
        volatility {float} -- Volatility of the underlying asset price.
        ttm {float} -- Time to expiration (in years).
        strike {float} -- Strike price of the option contract.
        rf {float} -- Risk-free rate (annual).

    Returns:
        float -- Price of a European Put Option contract.
    """

    d1, d2 = computeD1D2(current, volatility, ttm, strike, rf)

    put = (strike * np.exp(-1 * rf * ttm) * norm.cdf(-1 * d2)) \
        - (strike * norm.cdf(-1 * d1))

    return put
```

../fe621/black_scholes/put.py

2.1.2 Call Option

The Black-Scholes Option price for a European Call ($C(S_t)$) option is defined as:

$$C(S_t) = S_t\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2)$$

```python
from .util import computeD1D2

from scipy.stats import norm
import numpy as np


def blackScholesCall(current: float, volatility: float, ttm: float,
                     strike: float, rf: float) -> float:
    """Function to compute the Black-Scholes-Merton price of a European Call
    Option, parameterized by the current underlying asset price, volatility,
    time to expiration, strike price, and risk-free rate.

    Arguments:
        current {float} -- Current price of the underlying asset.
        volatility {float} -- Volatility of the underlying asset price.
        ttm {float} -- Time to expiration (in years).
        strike {float} -- Strike price of the option contract.
        rf {float} -- Risk-free rate (annual).
```

```
19
20      Returns:
21          float -- Price of a European Call Option contract.
22      """
23
24      d1, d2 = computeD1D2(current, volatility, ttm, strike, rf)
25
26      call = (current * norm.cdf(d1)) \
27          - (strike * np.exp(-1 * rf * ttm) * norm.cdf(d2))
28
29      return call
```

../fe621/black_scholes/call.py

2.1.3 Put-Call Parity

The relationship between the price of a Call and Put option is governed by Put-Call parity:

$$P(S_t) = C(S_t) - S_t + Ke^{-r(T-t)}$$

```
1  import numpy as np
2
3
4  def call(put: float, current: float, strike: float, ttm: float,
5           rf: float) -> float:
6      """Function to compute the price of a European Call option contract from a
7      European Put option contract price using Put-Call parity.
8
9      Arguments:
10         put {float} -- Price of the put option.
11         current {float} -- Current price of the underlying asset.
12         strike {float} -- Strike price of the option contract.
13         ttm {float} -- Time to expiration (in years).
14         rf {float} -- Risk-free rate (annual).
15
16     Returns:
17         float -- Price of a European Call Option contract.
18     """
19
20     return put + current - (strike * np.exp(-1 * rf * ttm))
21
22
23 def put(call: float, current: float, strike: float, ttm: float,
24         rf: float) -> float:
25     """Function to compute the price of a European Put option contract from a
26     European Call option contract price using Put-Call parity.
27
28     Arguments:
29         call {float} -- Price of the call option.
30         current {float} -- Current price of the underlying asset.
31         strike {float} -- Strike price of the option contract.
32         ttm {float} -- Time to expiration (in years).
33         rf {float} -- Risk-free rate (annual).
34
35     Returns:
36         float -- Price of a European Put Option contract.
37     """
38
39     return call - current + (strike * np.exp(-1 * rf * ttm))
```

../fe621/black_scholes/parity.py

2.1.4 The Greeks

The Greeks are the quantities representing the sensitivity of the price of a derivative with respect to changes in the underlying parameters. The following formulas are implemented to calculate each of the Greeks using the Black-Scholes option pricing formula. These formulas are derived in full in Stefanica 2011 and Weerawarana 2016.

***Note:* The following assumes the dividend rate, $q = 0$.**

Delta

The Delta (Δ) of an option is the first derivative of an option with respect to the price of the underlying asset at time t, S_t.

$$\Delta(C) = \frac{\partial C(S_t)}{\partial S_t} = \Phi(d_1)$$

Gamma

The Gamma (Γ) of an option is the second derivative of an option with respect to the price of the underlying asset at time t, S_t.

$$\Gamma(C) = \frac{\partial^2 C(S_t)}{\partial S_t^2} = \frac{\phi(d_1)}{S_t \sigma \sqrt{T - t}}$$

Vega

The Vega (ν) of an option is the first derivative of an option with respect to the volatility of the underlying asset at time t, σ.

$$\nu(C) = \nu(P) = \frac{\partial C(S_t)}{\partial \sigma} = S_t \sqrt{T - t}\, \phi(d_1)$$

```python
from .util import computeD1D2

from scipy.stats import norm

import numpy as np


def callDelta(current: float, volatility: float, ttm: float, strike: float,
              rf: float) -> float:
    """Function to compute the Delta of a call option using the Black-Scholes
    formula.

    Arguments:
        current {float} -- Current price of the underlying asset.
        volatility {float} -- Volatility of the underlying asset price.
        ttm {float} -- Time to expiration (in years).
        strike {float} -- Strike price of the option contract.
        rf {float} -- Risk-free rate (annual).
```

```
19
20      Returns:
21          float -- Delta of a European Call Option contract.
22      """
23
24      d1, d2 = computeD1D2(current, volatility, ttm, strike, rf)
25
26      return np.cdf(d1)
27
28
29  def callGamma(current: float, volatility: float, ttm: float, strike: float,
30                rf: float) -> float:
31      """Function to compute the Gamma of a Call option using the Black-Scholes
32      formula.
33
34      Arguments:
35          current {float} -- Current price of the underlying asset.
36          volatility {float} -- Volatility of the underlying asset price.
37          ttm {float} -- Time to expiration (in years).
38          strike {float} -- Strike price of the option contract.
39          rf {float} -- Risk-free rate (annual).
40
41      Returns:
42          float -- Delta of a European Call Opton Option contract.
43      """
44
45      d1, d2 = computeD1D2(current, volatility, ttm, strike)
46
47      return norm.pdf(d1) * (1 / (current * volatility * np.sqrt(ttm)))
48
49
50  def vega(current: float, volatility: float, ttm: float, strike: float,
51           rf: float) -> float:
52      """Function to compute the Vega of an option using the Black-Scholes formula.
53
54      Arguments:
55          current {float} -- Current price of the underlying asset.
56          volatility {float} -- Volatility of the underlying asset price.
57          ttm {float} -- Time to expiration (in years).
58          strike {float} -- Strike price of the option contract.
59          rf {float} -- Risk-free rate (annual).
60
61      Returns:
62          float -- Vega of a European Option contract.
63      """
64
65      d1, d2 = computeD1D2(current, volatility, ttm, strike, rf)
66
67      return current * np.sqrt(ttm) * norm.pdf(d1)
```

../fe621/black_scholes/greeks.py

2.2 Numeric Optimization

2.2.1 Bisection Method

In this section, we implement the Bisection optimization method. The bisection algorithm is outlined in Algorithm 1. The algorithm is implemented recursively.

Algorithm 1: Bisection Algorithm

Input: Input function, f to be optimized; must have sign change. Search space start and stop points, a and b. Tolerance level, ϵ.
Output: Point $x^* \in [a, b]$ where $f(x^*) = 0$.
Let midpont $= m$;
repeat
  $m = \frac{a+b}{2}$;
  if $f(a) \times f(mid) < 0$ **then**
    $b = m$
  end
  if $f(b) \times f(mid) < 0$ **then**
    $a = m$
  end
until $(b - a) < \epsilon$;
return $\frac{a+b}{2}$;

```python
from typing import Callable
import numpy as np


def bisectionSolver(f: Callable, a: float, b: float,
                    tol: float=10e-6) -> float:
    """Bisection method solver, implemented using recursion.

    Arguments:
        f {Callable} -- Function to be optimized.
        a {float} -- Lower bound.
        b {float} -- Upper bound.

    Keyword Arguments:
        tol {float} -- Solution tolerance (default: {10e-6}).

    Raises:
        Exception -- Raised if no solution is found.

    Returns:
        float -- Solution to the function s.t. f(x) = 0.
    """

    # Compute midpoint
    mid = (a + b) / 2

    # Check if estimate is within tolerance
    if (b - a) < tol:
        return mid

    # Evaluate function at midpoint
    f_mid = f(mid)
```

```
33
34      # Check position of estimate, move point and re-evaluate
35      if (f(a) * f_mid) < 0:
36          return bisectionSolver(f=f, a=a, b=mid)
37      elif (f(b) * f_mid) < 0:
38          return bisectionSolver(f=f, a=mid, b=b)
39      else:
40          raise Exception("No solution found.")
```

../fe621/optimization/bisection.py

2.2.2 Newton Method

In this section, we implement the Newton optimization method. The Newton method algorithm is outlined in Algorithm 2.

Algorithm 2: Newton's Method

Input: A differentiable function $f : \mathbb{R}^a \to \mathbb{R}^b \, \forall \, a, b \in \mathbb{N}_{>0}$. Starting guess for the root x_0. Tolerance level, ϵ.
Output: $x^* \in \mathbb{R}^a$, such that $f(x^*) = 0$
$k = 1$;
repeat
  $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$;
  $k = k + 1$
until $|f(x_k)| < \epsilon$;
return x_{k+1};

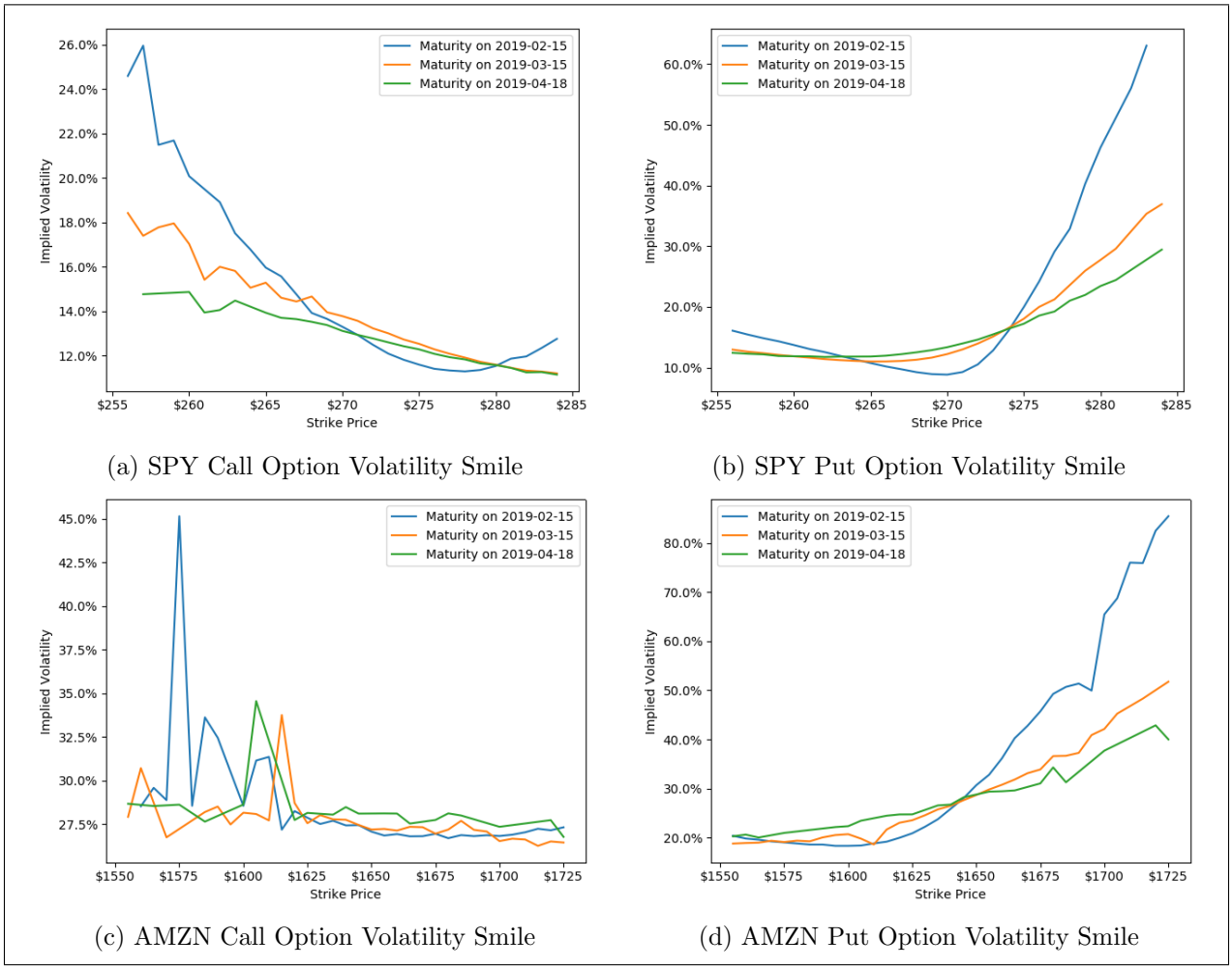
2.3 Implied Volatility

In this section, we utilize the functions and data described above to calculate the average implied volatility of each of the option chains. This was done for the entire dataset using the Bisection Method, but convergence times using the Newton Method were also explored.

2.4 Volatility Plots

2.4.1 Volatility Smile

2.4.2 Volatility Surface



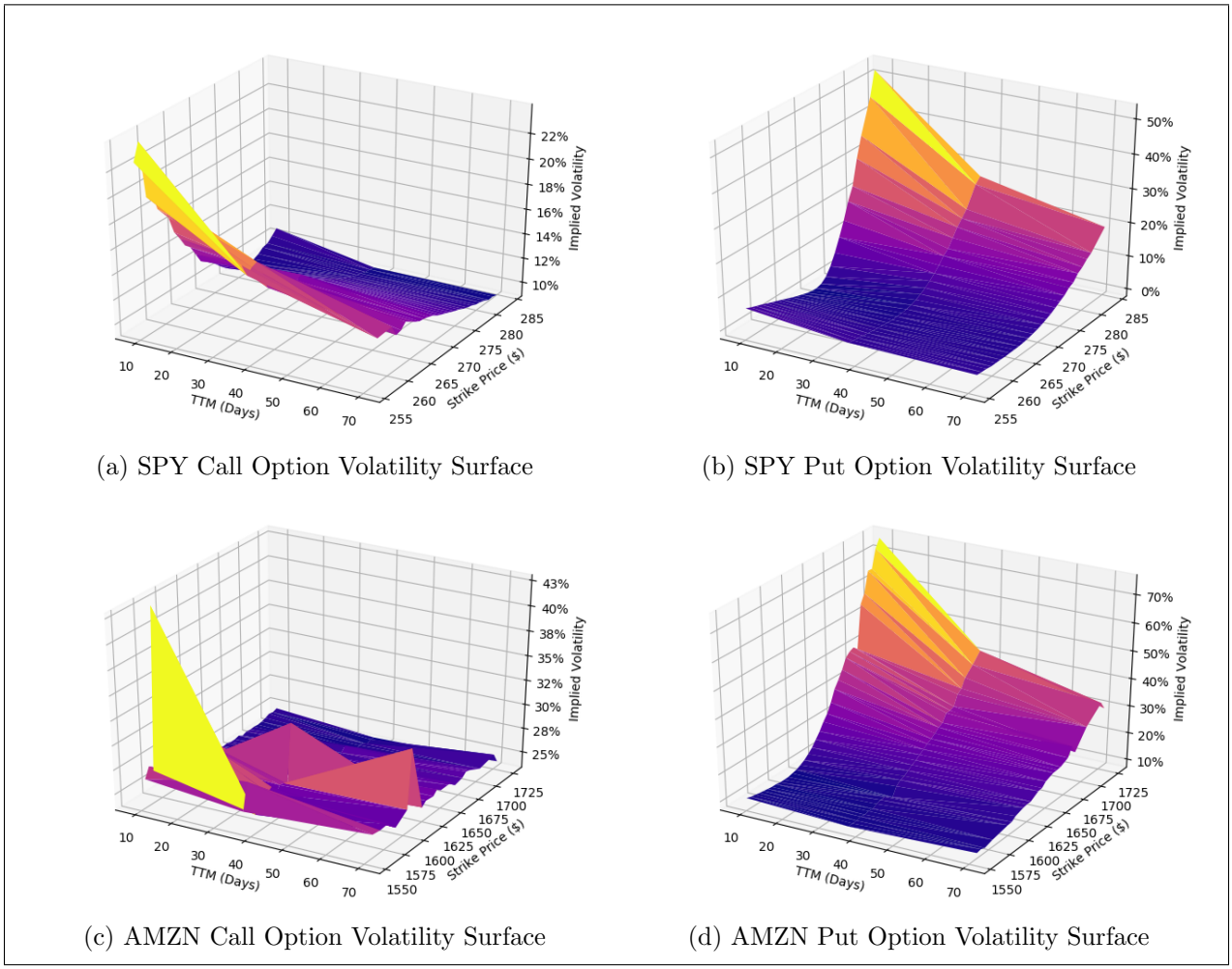
(a) SPY Call Option Volatility Smile

(b) SPY Put Option Volatility Smile

(c) AMZN Call Option Volatility Smile

(d) AMZN Put Option Volatility Smile

Figure 1: Volatility Smiles of call and put option chains on AMZN and SPY. Plots the relationship between the strike price and implied volatility for various maturities.



(a) SPY Call Option Volatility Surface

(b) SPY Put Option Volatility Surface

(c) AMZN Call Option Volatility Surface

(d) AMZN Put Option Volatility Surface

Figure 2: Volatility Surfaces of call and put option chains on AMZN and SPY. Plots the relationship between the strike price, time to maturity, and the implied volatility.

References

Armstrong, Whit, Dirk Eddelbuettel, and John Laing. 2018. *Rblpapi: R Interface to 'Bloomberg' (CRAN).* Accessed February 10, 2019. https://cran.r-project.org/web/packages/Rblpapi/index.html.

Bloomberg L.P. 2019. *Bloomberg Professional Services - Terminal.* New York, NY. https://www.bloomberg.com/professional/solution/bloomberg-terminal/.

Board of Governors of the Federal Reserve System. 2019. *Selected Interest Rates (Daily) - H.15.* Accessed February 12, 2019. https://www.federalreserve.gov/releases/h15/.

CBOE (Chicago Board Options Exchange). 2019. *VIX: Volatility Index.* Accessed February 12, 2019. http://www.cboe.com/vix.

Options Symbology Initiative Working Group. 2008. *Options Symbology Initiative.* Technical report. Chicago, IL: The Options Clearing Commission (OCC). https://www.theocc.com/components/docs/initiatives/symbology/symbology_initiative_v1_8.pdf.

Shreve, Steven E. 2004. *Stochastic Calculus for Finance II.* 153–164. April. Pittsburgh, PA: Springer Finance. ISBN: 0-387-40101-6.

State Street Global Advisors. 2019. *SPY: SPDR S&P 500 ETF Trust.* Accessed February 12, 2019. https://us.spdrs.com/en/etf/spdr-sp-500-etf-SPY.

Stefanica, Dan. 2011. *A Primer for the Mathematics of Financial Engineering.* First Edit. 89–96. New York, NY: FE Press. ISBN: 0-9797576-2-2.

Weerawarana, Rukmal. 2016. *Homework 3 - CFRM 460 (Mathematical Methods for Computational Finance) - University of Washington - rukmal - GitHub.* Accessed February 12, 2019. https://github.com/rukmal/CFRM-460-Homework/blob/master/Homework%203/Homework%203%20Solutions.pdf.

———. 2019. *FE 621 Homework - rukmal - GitHub.* Accessed February 20, 2019. https://github.com/rukmal/FE-621-Homework.

A Appendix

A.1 Question 1 Implementation

A.1.1 Bloomberg Data Download

```
library("Rblpapi")

# Connect to Bloomberg Terminal backend service
blpConnect(host = "localhost", port = 8194)


#----------------------------------
# Data Download Functionality
#----------------------------------


getPrice <- function(security, startTime, endTime, timeZone) {
  # Downloads and returns the closing price of a given security
  # for each minute in the trading day.
  #
  # Args:
  #   security: Name of the security to be downloaded.
  #   startTime: Datetime object with the start time.
  #   endTime: Datetime object with the end time.
  #   timeZone: Time zone of the target start and end times.
  #
  # Returns:
  #   DataFrame with the closing price for each minute in the
  #   trading day.

  # Getting price data
  data <- getBars(security = security, barInterval = 1,
                  startTime = startTime, endTime = endTime,
                  tz = timeZone)

  # Isolate time and closing price
  data <- data[c("times", "close")]

  # Rename columns
  colnames(data) <- c("Dates", "Close")

  # Return
  data
}


createOptionName <- function(security, dates, prices, type, suffix) {
  # Creates the Bloomberg-standard option name, given a security, date, price,
  # option type and suffix.
  #
  # Args:
  #   security: Name of the security to be included in the option price.
  #   dates: Dates to be included in option name.
  #   prices: Prices to be included in the option name.
  #   type: Type of the option ("C" or "P").
  #   suffix: Suffix for option name (typically "Index" or "Equity").
  #
  # Returns:
  #   Vector of Bloomberg-compatible option names.
```

```
55
56    # Empty vector to store names
57    names <- c()
58
59    # Iterate over each date and price
60    for (date in dates) {
61      for (price in prices) {
62        # Building option name
63        name <- paste(security, date, paste(type, price, sep = ""), suffix)
64
65        # Appending to list of option names
66        names <- c(names, name)
67      }
68    }
69
70    # Returning names
71    names
72  }
73
74
75  #---------------------------------
76  # DATA1
77  #---------------------------------
78
79
80  # Define Start and End times (DATA1)
81  data1Start <- ISOdatetime(year = 2019, month = 2, day = 6,
82                            hour = 9, min = 30, sec = 0)
83  data1End <- ISOdatetime(year = 2019, month = 2, day = 6,
84                          hour = 16, min = 0, sec = 0)
85
86  # Defining time zone
87  timeZone = "America/New_York"
88
89  # Defining top-level securities
90  securities <- c("SPY US Equity", "AMZN US Equity", "VIX Index")
91
92  # Getting prices for each of the top-level securities
93  for (security in securities) {
94    data <- getPrice(security, data1Start, data1End, timeZone)
95    write.csv(data, file = paste(security, "DATA1", "csv", sep = "."),
96              row.names = FALSE)
97  }
98
99  # Expiration dates
100 expDates <- c("2/15/19", "3/15/19", "4/18/19")
101
102 # Defining put and call prices for SPY and AMZN options
103 # Grabbing prices for 5% +/- current price
104
105 # Current SPY price
106 spyCurrent <- 270
107 spyPrices <- c(floor(0.95 * spyCurrent):ceiling(1.05 * spyCurrent))
108
109 # Current AMZN price (need to do this manually because of option strikes)
110 # Closest option price to 95% of price is at $1557.50 and 105% is $1722.50
111 amznCurrent <- 1640
112 amznPrices <- seq(1555, 1725, by=5)
113
114 # Creating option names for SPY and AMZN
115 spyOptions <- createOptionName("SPY", expDates, spyPrices, "C", "Equity")
```

```r
spyOptions <- c(spyOptions, createOptionName("SPY", expDates, spyPrices,
                                             "P", "Equity"))

amznOptions <- createOptionName("AMZN", expDates, amznPrices, "C", "Equity")
amznOptions <- c(amznOptions, createOptionName("AMZN", expDates, amznPrices,
                                               "P", "Equity"))

# Getting prices for each of the options
for (option in c(amznOptions, spyOptions)) {
  data <- getPrice(option, data1Start, data1End, timeZone)
  # Only print to file if option exists
  if (all(dim(data) > 0)) {
    optionFileName <- gsub("/", "-", option) # Need to do this for Windows
    write.csv(data, file = paste(optionFileName, "csv", sep = "."),
              row.names = FALSE)
  }
}


#----------------------------------
# DATA2
#----------------------------------

# Define Start and End times (DATA2)
data2Start <- ISOdatetime(year = 2019, month = 2, day = 7,
                          hour = 9, min = 30, sec = 0)
data2End <- ISOdatetime(year = 2019, month = 2, day = 7,
                        hour = 16, min = 0, sec = 0)

# Getting prices for each of the top-level securities
for (security in securities) {
  data <- getPrice(security, data2Start, data2End, timeZone)
  write.csv(data, file = paste(security, "DATA2", "csv", sep = "."),
            row.names = FALSE)
}
```

question_solutions/question_1.R

A.1.2 Data Cleaning

```python
# Script to rename option files, from the R data download script format to
# OOC-compliant names.

from context import fe621

import os

option_file_paths = [os.getcwd() + i for i in ['/Homework 1/data/DATA1/AMZN',
                                               '/Homework 1/data/DATA1/SPY']]

for option_file_path in option_file_paths:
    fe621.util.renameOptionFiles(folder_path=option_file_path)
```

data_cleaning.py