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[1]. Using this, we calculate and 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 B for specific question implementations, and the project GitHub repository[2] for full source code of the `fe621` Python package.

1. Shreve 2004
2. Weerawarana 2019

1 Data Overview

1.1 Asset Descriptions

1.1.1 *SPY* - SPDR S&P 500 ETF[3]

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[4]

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**).

3. State Street Global Advisors 2019
4. CBOE (Chicago Board Options Exchange) 2019

This data detailed above was gathered utilizing *Rblpapi*[5], which provides an R interface to data on the Bloomberg Terminal[6]. 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 B.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.[7]

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[8].

1.3 Option Naming Convention

A modern convention for naming option contracts was proposed by the Options Clearing Commission (OCC) in 2008[9], 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:

5. Armstrong et al. 2018
6. Bloomberg L.P. 2019
7. Board of Governors of the Federal Reserve System 2019
8. Weerawarana 2019
9. Options Symbology Initiative Working Group 2008

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 $\rightarrow$ 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`[10] *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

10. Weerawarana 2019

***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
```

```python
    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 Call Option contract.
    """

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

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

    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:

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

```python
import numpy as np


def call(put: float, current: float, strike: float, ttm: float,
         rf: float) -> float:
    """Function to compute the price of a European Call option contract from a
    European Put option contract price using Put-Call parity.

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

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

    return put + current - (strike * np.exp(-1 * rf * ttm))


def put(call: float, current: float, strike: float, ttm: float,
        rf: float) -> float:
    """Function to compute the price of a European Put option contract from a
    European Call option contract price using Put-Call parity.

    Arguments:
        call {float} -- Price of the call option.
        current {float} -- Current price of the underlying asset.
```

```
        strike {float} -- Strike price of the option contract.
        ttm {float} -- Time to expiration (in years).
        rf {float} -- Risk-free rate (annual).

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

    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.

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

```
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).

    Returns:
        float -- Delta of a European Call Option contract.
    """

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

    return np.cdf(d1)


def callGamma(current: float, volatility: float, ttm: float, strike: float,
              rf: float) -> float:
    """Function to compute the Gamma 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).

    Returns:
        float -- Delta of a European Call Opton Option contract.
    """

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

    return norm.pdf(d1) * (1 / (current * volatility * np.sqrt(ttm)))


def vega(current: float, volatility: float, ttm: float, strike: float,
         rf: float) -> float:
    """Function to compute the Vega of an 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).

    Returns:
        float -- Vega of a European Option contract.
    """

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

    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 midpoint $= m$;
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)
```

```python

    # Check position of estimate, move point and re-evaluate
    if (f(a) * f_mid) < 0:
        return bisectionSolver(f=f, a=a, b=mid)
    elif (f(b) * f_mid) < 0:
        return bisectionSolver(f=f, a=mid, b=b)
    else:
        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.[11]

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 $|x_k - x_{k-1}| < \epsilon$;
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.

2.3.1 Convergence Comparison

Here, we compare the performance of each of the optimization methods described above, the Bisection method and Newton method. This was done by computing the average daily implied volatility on the complete SPY option chain in the dataset.

The average daily implied volatility is computed by first calculating the implied volatility by-minute. Then, the mean of these minute-level implied volatilities is computed and is treated as the average daily implied volatility of the given option. For this comparison, the tolerance level of each of the termination conditions was set to 1×10^{-4}.

The time elapsed for these computations, and other related statistics under each of the two optimization methods are presented in Table 1.

Despite having a theoretical quadratic convergence rate, Newton's method results in slower performance compared to the Bisection method. This is evident from both the total time elapsed, and the average time per operation (computed to include dropped option computations for consistency).

11. Stefanica 2011

	Newton Method	Bisection Method
Number of Input Options	165.0	165.0
Number of Output Options	164.0	164.0
Number of Dropped Options	1.0	1.0
Time Elapsed for Computation (s)	2423.1484701633453	2406.0394039154053
Average Time per Option (s)	14.685748304020274	14.582056993426699

Table 1: Convergence comparison of average daily implied volatility computation on the SPY option chain using the Bisection and Newton optimization methods.

This can be attributed to the fact that some of the minute-level implied volatility optimizations do not have solutions. The Bisection method reaches a state of "no solution" faster than Newton's method, as it employs a technique of reducing the possible range of the solution. This converging search space would suggest it discovers a state of "no solution" faster than the unbounded search space of the Newton method. In principle - on the condition that the existence of a solution is guaranteed - the Newton method will converge faster than the Bisection method, given a reasonable initial guess.

2.3.2 Average Daily Implied Volatility

Average daily implied volatility was computed for each option, across all strike prices and expiration dates, for both SPY and AMZN option chains. This optimization on the aggregate dataset was completed using the Bisection Method.

This was done by first computing the implied volatility for each minute, solving for some σ such that $(C(S_t)|_\sigma - P = 0)$ or $(P(S_t)|_\sigma - P = 0)$ for a call or put option respectively. Then, the mean of each of these implied volatilities was computed to obtain the daily average implied volatility for an option with a given strike price and expiration date. For this comparison, the tolerance level of each of the termination conditions was set to 1×10^{-7}.

The complete dataset of average daily implied volatility is reproduced for the complete option chains on SPY in Appendix A.1 and AMZN in Appendix A.2.

2.3.3 *Money-ness* Implied Volatility Comparison

We also compared the average daily implied volatility of options *in-the-money*, and *out-of-the-money*. For this comparison, we defined the ratio of *money-ness* to be $\pm 5\%$ of the current underlying asset price, where options within the range are *in-the-money*, and *out-of-the-money* otherwise. This comparison data is presented in Table 2.

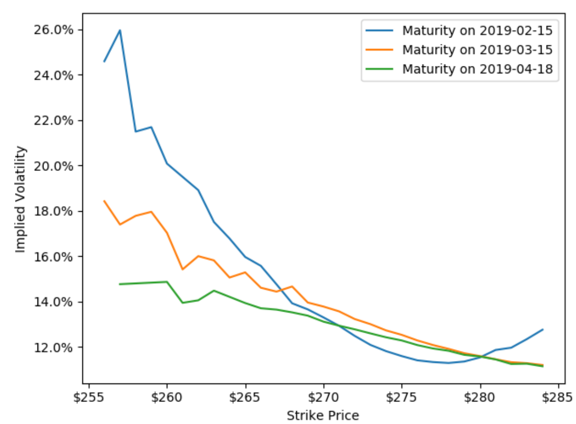
	SPY	AMZN
In-the-money Options Average Daily Implied Vol	0.15739310934145576	0.29781477676343643
Out-of-the-money Options Average Daily Implied Vol	0.16631389625540363	0.3189350623328305

Table 2: Comparison of *in-the-money* and *out-of-the-money* options through the lens of their average daily implied volatility.

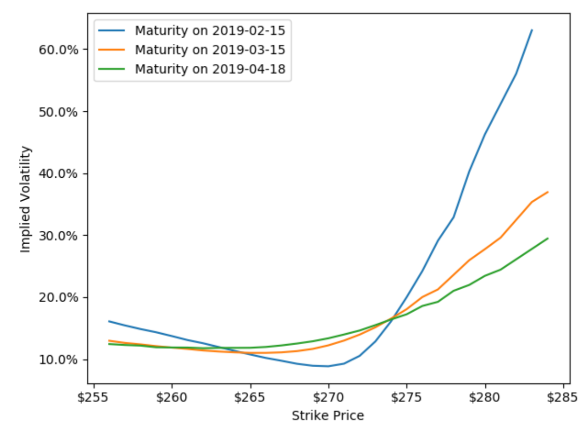
2.4 Volatility Plots

2.4.1 Volatility Smile

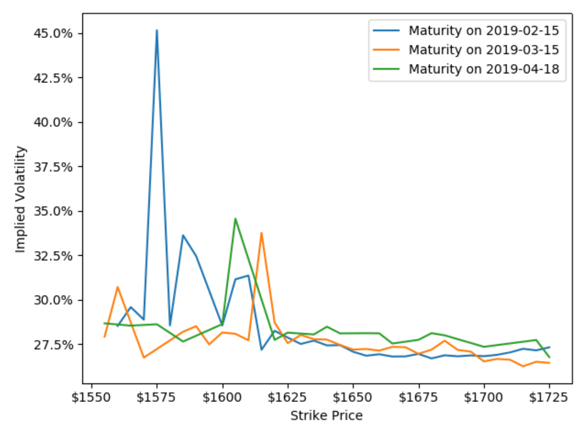
2.4.2 Volatility Surface



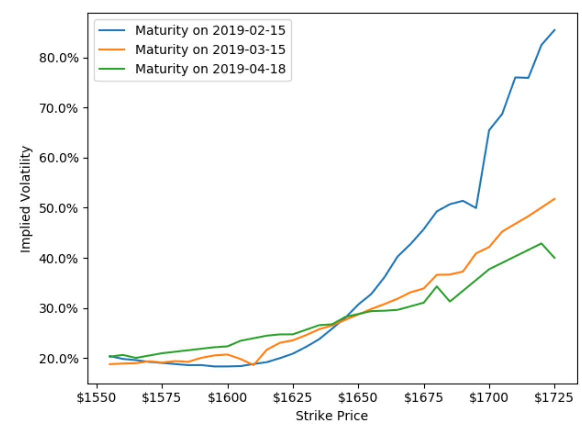
(a) SPY Call Option Volatility Smile



(b) SPY Put Option Volatility Smile

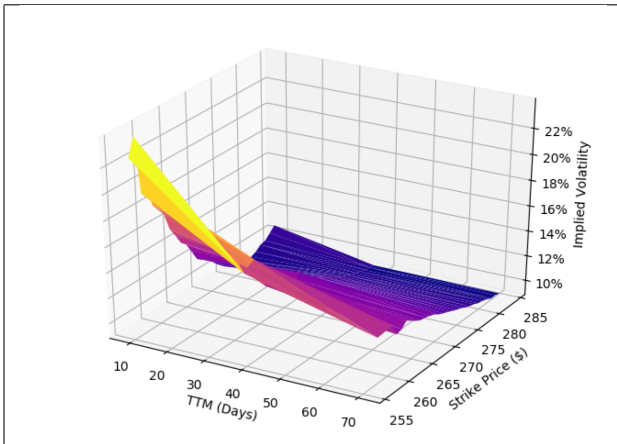


(c) AMZN Call Option Volatility Smile

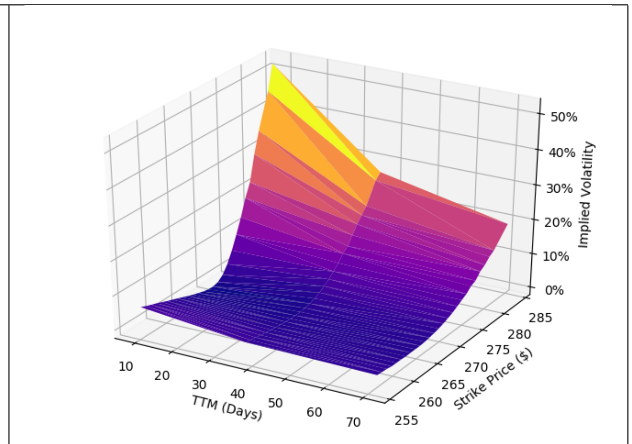


(d) AMZN Put Option Volatility Smile

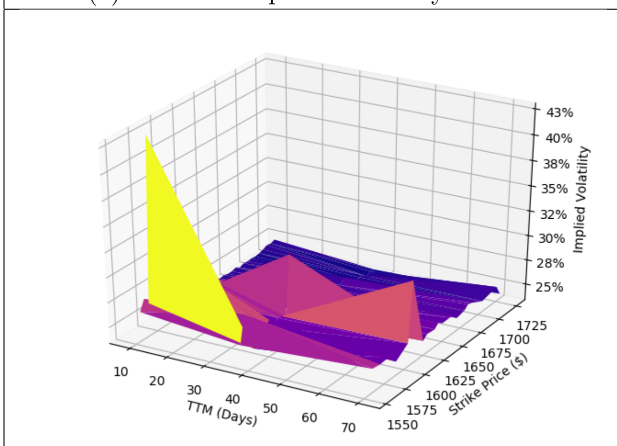
Figure 1: Volatility Smiles of call and put option chains on AMZN and SPY. Plots the relationship between the strike price and implied volatility for various maturities.



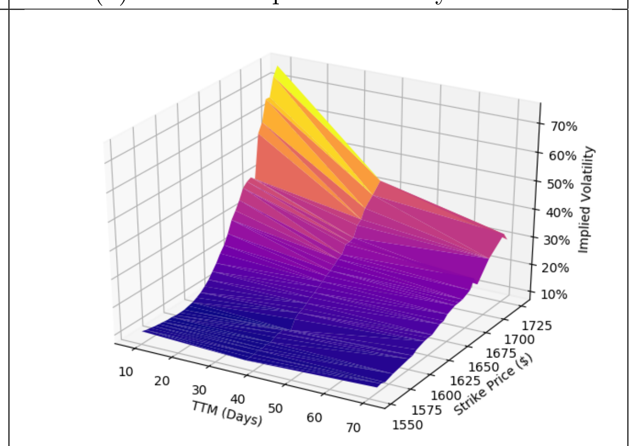
(a) SPY Call Option Volatility Surface



(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.

3 Numerical Integration

3.1 Quadrature Methods

In this section, we implement the *Trapezoidal Rule* and *Simpson's Rule* quadrature methods.

$$\text{Let data} = \boldsymbol{x}$$
$$\text{Let } i^{\text{th}} \text{ element of } \boldsymbol{x} = x_i$$

3.1.1 Trapezoidal Rule

$$\text{Let Trapezoidal rule approximation} = T_N(f)$$
$$\Rightarrow T_N(f) = \sum_{i=1}^{N}\left[\left(\frac{f(x_{i-1}) + f(x_i)}{2}\right) \times h\right]$$
$$\Rightarrow h \times \sum_{i=1}^{N}\left[\frac{f(x_{i-1}) + f(x_i)}{2}\right] = h \times \left(\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{N-1} + \frac{1}{2}f(x_N))\right)$$
$$\therefore T_N(f) = hf(\boldsymbol{x}) - \frac{h}{2}(f(x_0) + f(x_N))$$

```
from typing import Callable
import numpy as np


def trapezoidalRule(f: Callable, N: float, start: float=-1e6,
                    stop: float=1e6) -> float:
    """Function to approximate the numeric integral of a function, f, using
    the Trapezoidal rule.

    Arguments:
        f {Callable} -- Function who's integral is to be estimated.
        N {int} -- Number of nodes to consider.

    Keyword Arguments:
        start {float} -- Starting point (default: {-1e6}).
        stop {float} -- Stopping point (default: {1e6}).

    Returns:
        float -- Approximation of the area under the function.
    """

    # Building values for approximation, and getting step size
    x, h = np.linspace(start=start, stop=stop, num=N, retstep=True)

    # Estimating area using trapezoidal rule, return
    return np.sum((h * f(x))) - ((h / 2) * (f(start) + f(stop)))
```

../fe621/numerical_integration/trapezoidal.py

3.1.2 Simpson's Rule

The following equation is derived in full in (Florescu 2019).

$$\text{Let Simpson's rule approximation} = S_N(f)$$
$$\Rightarrow S_N(f) \approx \frac{h}{6} \times \sum_{i=1}^{N} \left[f(x_{i-1}) + 4f\left(\frac{x_{i-1}+x_i}{2}\right) + f(x_i) \right]$$
$$= \frac{h}{6}\left(\sum_{i=1}^{N} [f(x_{i-1}) + f(x_i)] + 4 \times \sum_{i=1}^{N} \left[f\left(\frac{x_{i-1}+x_i}{2}\right)\right]\right)$$

$$\text{Note that } \left(\frac{x_{i-1}+x_i}{2}\right) \text{ is the midpoint between the points in } \boldsymbol{x}.$$
$$\text{Let the above} = \boldsymbol{x}_{\text{mid}}$$
$$\therefore\ S_N(f) \approx \frac{h}{6}\left(2f(\boldsymbol{x}) - (f(x_0) + f(x_N)) + 4f(\boldsymbol{x}_{\text{mid}})\right)$$

```
from typing import Callable
import numpy as np


def simpsonsRule(f: Callable, N: float, start: float=-1e6,
                 stop: float=1e6) -> float:
    """Function to approximate the numeric integral of a function, f, using
    Simpson's rule.

    Arguments:
        f {Callable} -- Function for which the integral is to be estimated.
        N {float} -- Number of nodes to consider.

    Keyword Arguments:
        start {float} -- Starting point (default: {-1e6}).
        stop {float} -- Stopping point (default: {1e6}).

    Returns:
        float -- Approximation of the area under the function.
    """

    # Building values for approximation, and getting step size
    x, h = np.linspace(start=start, stop=stop, num=N, retstep=True)

    # Computing midpoints
    x_mid = np.array([(x[i - 1] + x[i]) / 2 for i in range(1, N)])

    # Estimating using Simpson's rule
    area = np.sum(2 * f(x)) - (f(start) + f(stop)) + (4 * np.sum(f(x_mid)))

    # Scaling area
    area *= (h / 6)

    return area
```

../fe621/numerical_integration/simpsons.py

3.2 Truncation Error Analysis

To examine the behavior of each of the quadrature methods described above, we approximate the integral of the following function:

$$f(x) = \begin{cases} \frac{\sin(x)}{x}, & \text{for } x \neq 0, \\ 1, & \text{for } x = 0. \end{cases}$$

We parameterize the start and stop points of the quadrature methods with a variable a, such that start $= -a$ and stop $= a$. Furthermore, we define the number of segments with variable N.

$$\text{Let approximation with parameters } a \text{ and } N = I_{N,a}$$

It is know analytically that the value of the integral $\int_{\infty}^{\infty} f(x)dx = \pi$. We evaluate the performance of each of the quadrature methods with various values of a and N. Then, we compute the *truncation error* of the approximation, defined as:

$$\text{Truncation error for approximation with parameters } a \text{ and } N = |I_{N,a} - \pi|$$

	N = 1000	N = 10000	N = 100000	N = 1000000	N = 10000000
a = 100	1.70837393e-02	7.23666391e-04	6.28030676e+00	6.28753820e+00	8.02007982e-04
a = 1000	1.71411414e-02	1.12266273e-03	1.22268346e-04	6.28287768e+00	6.28285876e+00
a = 10000	1.71417140e-02	1.12637221e-03	1.89801996e-04	1.28334824e-05	6.28321420e+00
a = 100000	1.71417198e-02	1.12640928e-03	1.90430835e-04	1.99205411e-05	1.20297055e-06
a = 1000000	1.71417198e-02	1.12640965e-03	1.90437119e-04	1.99865427e-05	1.86725626e-06

Table 3: Trapezoidal quadrature rule truncation error for varying values of a and N.

	N = 1000	N = 10000	N = 100000	N = 1000000	N = 10000000
a = 100	1.71417296e-02	1.13348528e-03	1.04706334e+01	1.27753456e+02	1.33203419e+03
a = 1000	1.71417198e-02	1.12641028e-03	1.91636193e-04	1.04718335e+01	1.27758461e+02
a = 10000	1.71417198e-02	1.12640965e-03	1.90437288e-04	2.01130216e-05	1.04719888e+01
a = 100000	1.71417198e-02	1.12640965e-03	1.90437182e-04	1.99872209e-05	1.88530816e-06
a = 1000000	1.71417198e-02	1.12640965e-03	1.90437182e-04	1.99872089e-05	1.87350648e-06

Table 4: Simpsons quadrature rule truncation error for varying values of a and N.

Table 3 and Table 4 report the truncation error for the Trapezoidal and Simpson's quadrature rules, respectively. The script used to produce this table is reproduced in Appendix B.3.1. Variations of N and a are explores in increasing powers of 10, with a progressing from 100 to 1,000,000, and N from 1,000 to 10,000,000.

It is evident from Table 3 that the Trapezoidal quadrature rule performs relatively well across all values of a, even at relatively low values of N. Compared to Simpson's quadrature rule truncation error (Table 4), the Trapezoidal quadrature rule also performs relatively better with larger values of N, and small values of a.

A potential explanation of this may be the interpolating behavior of the Simpson's quadrature rule. The function $\frac{sin(x)}{x}$ is significantly more linear than quadratic in small intervals, and thus the quadratic interpolating behavior of the Simpson's quadrature rule is a poor approximation heuristic for the function with low values of a.

Finally, it can be observed that both quadrature rule approximations converge commensurately as the values of a and N increase. However, it is clear that the Trapezoidal quadrature rule approximation converges at a faster rate than the Simpson's quadrature rule approximation with increasing values of a and N.

3.3 Convergence Analysis

Typically, the true value of the objective integral is unknown. In this case, we would evaluate the rate of change of the objective function (i.e. convergence) computation with respect to the number of segments, N. We assign an arbitrary convergence criteria, ϵ to test the convergence with progressively increasing (in powers of 10) values of N.

$$\text{Let approximation with parameter } N = I_N$$
$$\text{Repeat while: } |I_N - I_{N_{\text{old}}}| > \epsilon$$

We evaluate the number of iterations required for a convergence level of $\epsilon = 10^{-3}$ for the Trapezoidal and Simpson's quadrature rules. The output of this evaluation is reproduced in Table 5. The solution source code for this analysis is reproduced in Appendix B.3.2. The `fe-621` package[12] sub-module used in this analysis is presented below.

```
from typing import Callable, Tuple
import numpy as np


def convergenceApproximation(f: Callable, rule: Callable, epsilon: float=1e-3,
                             start: float=-1e6, stop: float=1e6) \
                            -> Tuple[float, int]:
    """Function to approximate the numeric integral of a function, f, using
    a given quadrature rule and a tolerance level epsilon.

    Arguments:
        f {Callable} -- Function for which the integral is to be estimated.
        rule {Callable} -- Function to be used to approximate area. Must take
                           positional arguments f, N, start and stop.

    Keyword Arguments:
        epsilon {float} -- Tolerance level (default: {1e-3}).
        start {float} -- Starting point (default: {-1e6}).
        stop {float} -- Stopping point (default: {1e6}).

    Returns:
        Tuple[float, int] -- Approximation of the area under the function
                             and the number of segments (area, segments).
    """

    # Flags
    area_old = 0
    area_new = 1
    N = 1

    while (np.abs(area_new - area_old) > epsilon):
        # Set new area to old area
        area_old = area_new
```

12. Weerawarana 2019

```python

        # Increase N by powers of 10
        N *= 10

        # Computing area with given parameters
        area_new = rule(f=f, N=N, start=start, stop=stop)

        # Log
        print('On iteration {0} method {1} convergence {2} val {3}'.format(
            N, str(rule),
            '{:.5e}'.format(np.abs(area_new - area_old)),
            area_new))

    # Return final area and number of segments
    return (area_new, N)
```

../fe621/numerical_integration/convergence.py

	Estimated Area	Segments
Trapezoidal Rule	3.14162154e+00	1.00000000e+05
Simpson's Rule	3.14159078e+00	1.00000000e+07

Table 5: Analysis of segments required for convergence under the Trapezoidal and Simpson's quadrature rules.

Analyzing the results in Table 5, it is evident that the number of segments required for convergence under the Trapezoidal quadrature rule is significantly less than that required under Simpson's quadrature rule. This difference is significant, with the Trapezoidal quadrature rule requiring segments two orders of magnitude less than Simpson's quadrature rule for convergence. These behavior is in agreement with the previous analysis of convergence with respect to varying values of N and a, explored in Section 3.2.

3.3.1 Arbitrary Function

Additionally, we also evaluate each quadrature rule with respect to the number of segments required for convergence with an additional arbitrary integral:

$$g(x) = 1 + e^{-x^2} \cos\left(8x^{\frac{2}{3}}\right)$$
$$\int_0^2 g(x)\,dx$$

	Estimated Area	Segments
Trapezoidal Rule	1.95879798e+00	1.00000000e+04
Simpson's Rule	1.95879793e+00	1.00000000e+03

Table 6: Analysis of segments required for convergence of an arbitrary integral under the Trapezoidal and Simpson's quadrature rules.

The estimates and segments required for convergence for the integral $\int_0^2 g(x)\,dx$ are presented in Table 6. The source code for this analysis is reproduced in Appendix B.3.3.

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`.

Florescu, Ionut. 2019. "1.11.4 Simpson's Rule." Chap. 1 in *Computational Methods in Finance,* 25–26. Hoboken, NJ.

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 Computed Implied Volatility

A.1 SPY Option Chain

Option Name	Expiration Date	Type	Strike	Implied Volatility
SPY190418P00284000	2019-04-18	P	284.0	0.2944151519814416
SPY190215C00273000	2019-02-15	C	273.0	0.12088632949477876
SPY190315P00280000	2019-03-15	P	280.0	0.2773567965573362
SPY190315C00259000	2019-03-15	C	259.0	0.1795022993746316
SPY190215P00282000	2019-02-15	P	282.0	0.5605288234818012
SPY190418C00279000	2019-04-18	C	279.0	0.11649886665441801
SPY190315C00271000	2019-03-15	C	271.0	0.1356402809357704
SPY190418C00267000	2019-04-18	C	267.0	0.13641568400975687
SPY190215C00257000	2019-02-15	C	257.0	0.2594671356544066
SPY190315C00263000	2019-03-15	C	263.0	0.1580963842094402
SPY190418C00275000	2019-04-18	C	275.0	0.12282794698729844
SPY190418P00265000	2019-04-18	P	265.0	0.11821970000596302
SPY190215P00259000	2019-02-15	P	259.0	0.14327652923896184
SPY190315P00273000	2019-03-15	P	273.0	0.15124846602339878
SPY190215C00280000	2019-02-15	C	280.0	0.11533855477257458
SPY190418P00277000	2019-04-18	P	277.0	0.19241970823244062
SPY190315P00261000	2019-03-15	P	261.0	0.11638052323285271
SPY190418P00269000	2019-04-18	P	269.0	0.12876836235261024
SPY190315P00257000	2019-03-15	P	257.0	0.12613319679904167
SPY190215P00263000	2019-02-15	P	263.0	0.11941925643959923
SPY190315C00282000	2019-03-15	C	282.0	0.11325154463043603
SPY190215P00271000	2019-02-15	P	271.0	0.09264454512340028
SPY190315P00263000	2019-03-15	P	263.0	0.1122342107241111
SPY190215P00257000	2019-02-15	P	257.0	0.15432878528409602
SPY190418P00275000	2019-04-18	P	275.0	0.17227954571814183
SPY190315P00271000	2019-03-15	P	271.0	0.1299093080603558
SPY190418P00279000	2019-04-18	P	279.0	0.21964454894785382
SPY190215C00282000	2019-02-15	C	282.0	0.11963784542230084
SPY190418P00267000	2019-04-18	P	267.0	0.12203147039389062
SPY190315C00280000	2019-03-15	C	280.0	0.11594249159478776
SPY190215P00273000	2019-02-15	P	273.0	0.1284050392677717
SPY190315P00259000	2019-03-15	P	259.0	0.12089371986096473
SPY190215P00261000	2019-02-15	P	261.0	0.13070634563865563
SPY190418C00284000	2019-04-18	C	284.0	0.11145753933645575
SPY190215C00271000	2019-02-15	C	271.0	0.1292855172510952
SPY190215C00263000	2019-02-15	C	263.0	0.17502479961523587
SPY190315C00257000	2019-03-15	C	257.0	0.17389593831718425
SPY190418C00277000	2019-04-18	C	277.0	0.11929305922954589
SPY190418C00269000	2019-04-18	C	269.0	0.13374824962957435
SPY190315C00261000	2019-03-15	C	261.0	0.15413590404383667
SPY190215C00259000	2019-02-15	C	259.0	0.21678821713316673
SPY190418C00265000	2019-04-18	C	265.0	0.13930426839062626

Option Name	Expiration Date	Type	Strike	Implied Volatility
SPY190215P00280000	2019-02-15	P	280.0	0.4625415192235766
SPY190315C00273000	2019-03-15	C	273.0	0.1300222367581809
SPY190215P00265000	2019-02-15	P	265.0	0.10747603443272584
SPY190418P00259000	2019-04-18	P	259.0	0.119044835610158
SPY190418C00280000	2019-04-18	C	280.0	0.11576322033582136
SPY190215P00277000	2019-02-15	P	277.0	0.2910963165790529
SPY190315C00284000	2019-03-15	C	284.0	0.11202337796730763
SPY190215P00269000	2019-02-15	P	269.0	0.08925885495627323
SPY190315P00275000	2019-03-15	P	275.0	0.18053468231045072
SPY190418P00263000	2019-04-18	P	263.0	0.1180703072901577
SPY190315P00267000	2019-03-15	P	267.0	0.11089874960272514
SPY190418P00271000	2019-04-18	P	271.0	0.13969507973517298
SPY190315P00279000	2019-03-15	P	279.0	0.2595404285908965
SPY190315C00269000	2019-03-15	C	269.0	0.1395301501769239
SPY190418C00261000	2019-04-18	C	261.0	0.13939978216615173
SPY190315C00277000	2019-03-15	C	277.0	0.12082013327752233
SPY190418C00273000	2019-04-18	C	273.0	0.12595034011489595
SPY190315C00265000	2019-03-15	C	265.0	0.15280866257065093
SPY190215C00279000	2019-02-15	C	279.0	0.11354740318434928
SPY190215C00267000	2019-02-15	C	267.0	0.1475665514426463
SPY190418C00257000	2019-04-18	C	257.0	0.14762643048220583
SPY190215C00275000	2019-02-15	C	275.0	0.11595551620054123
SPY190315C00267000	2019-03-15	C	267.0	0.14434518106758137
SPY190315C00279000	2019-03-15	C	279.0	0.11721956150611039
SPY190418C00271000	2019-04-18	C	271.0	0.12932551791295982
SPY190315C00275000	2019-03-15	C	275.0	0.12531296371498987
SPY190418C00263000	2019-04-18	C	263.0	0.14476057818478635
SPY190315P00284000	2019-03-15	P	284.0	0.3692597318488314
SPY190215C00277000	2019-02-15	C	277.0	0.11332081406927474
SPY190215C00269000	2019-02-15	C	269.0	0.1365114173011097
SPY190215C00265000	2019-02-15	C	265.0	0.1595995284482182
SPY190418P00280000	2019-04-18	P	280.0	0.23431608439101587
SPY190418P00257000	2019-04-18	P	257.0	0.12276099465997017
SPY190215P00275000	2019-02-15	P	275.0	0.19981958677091866
SPY190215P00279000	2019-02-15	P	279.0	0.40279262815899863
SPY190418C00282000	2019-04-18	C	282.0	0.11246448282695487
SPY190215P00267000	2019-02-15	P	267.0	0.09729181714070118
SPY190418P00273000	2019-04-18	P	273.0	0.15478764653510754
SPY190315P00265000	2019-03-15	P	265.0	0.11004800381867783
SPY190418P00261000	2019-04-18	P	261.0	0.11858248649655706
SPY190315P00269000	2019-03-15	P	269.0	0.11642764596378102
SPY190215C00284000	2019-02-15	C	284.0	0.12756590343192412
SPY190315P00277000	2019-03-15	P	277.0	0.21241694155251584
SPY190215P00262000	2019-02-15	P	262.0	0.12552745506891508
SPY190315P00256000	2019-03-15	P	256.0	0.12959581506831566
SPY190215P00270000	2019-02-15	P	270.0	0.08841481660028248

Option Name	Expiration Date	Type	Strike	Implied Volatility
SPY190315C00283000	2019-03-15	C	283.0	0.1128300986326564
SPY190215C00281000	2019-02-15	C	281.0	0.11863212146417564
SPY190315P00272000	2019-03-15	P	272.0	0.13956624833519196
SPY190215P00258000	2019-02-15	P	258.0	0.14846573705258576
SPY190418P00264000	2019-04-18	P	264.0	0.11810942988871309
SPY190418P00268000	2019-04-18	P	268.0	0.1251350095509873
SPY190315P00260000	2019-03-15	P	260.0	0.11850736330232352
SPY190418P00276000	2019-04-18	P	276.0	0.18565303529314983
SPY190418C00266000	2019-04-18	C	266.0	0.13701354756074793
SPY190315C00270000	2019-03-15	C	270.0	0.13776912103833444
SPY190418C00278000	2019-04-18	C	278.0	0.11827662777717766
SPY190215P00283000	2019-02-15	P	283.0	0.6304874200650188
SPY190418C00274000	2019-04-18	C	274.0	0.12422406764896325
SPY190315C00262000	2019-03-15	C	262.0	0.1599738542990916
SPY190215C00256000	2019-02-15	C	256.0	0.24582996895785705
SPY190215C00260000	2019-02-15	C	260.0	0.20072617344350122
SPY190315C00258000	2019-03-15	C	258.0	0.1777208796547502
SPY190315P00281000	2019-03-15	P	281.0	0.29590371319704956
SPY190215C00272000	2019-02-15	C	272.0	0.12483576069707455
SPY190315C00260000	2019-03-15	C	260.0	0.1702547805083682
SPY190418C00268000	2019-04-18	C	268.0	0.13520112732792144
SPY190418C00276000	2019-04-18	C	276.0	0.12080398666889161
SPY190315C00272000	2019-03-15	C	272.0	0.13226115185281503
SPY190215C00258000	2019-02-15	C	258.0	0.2148171833583287
SPY190315P00283000	2019-03-15	P	283.0	0.35343907983101847
SPY190215C00270000	2019-02-15	C	270.0	0.13298826144479425
SPY190315C00256000	2019-03-15	C	256.0	0.1841575959149529
SPY190215C00262000	2019-02-15	C	262.0	0.18904398203591216
SPY190315P00258000	2019-03-15	P	258.0	0.12373132778860418
SPY190215P00272000	2019-02-15	P	272.0	0.10526036362513862
SPY190315C00281000	2019-03-15	C	281.0	0.11454295624247597
SPY190215P00260000	2019-02-15	P	260.0	0.1371335251556943
SPY190418P00274000	2019-04-18	P	274.0	0.1638658699172232
SPY190215P00256000	2019-02-15	P	256.0	0.16078886778458304
SPY190315P00262000	2019-03-15	P	262.0	0.11395265379220323
SPY190418P00266000	2019-04-18	P	266.0	0.11957399070720233
SPY190215C00283000	2019-02-15	C	283.0	0.12342878619728186
SPY190418P00278000	2019-04-18	P	278.0	0.2101958804118359
SPY190315P00270000	2019-03-15	P	270.0	0.12220596108595123
SPY190215C00266000	2019-02-15	C	266.0	0.15568796054337375
SPY190215C00278000	2019-02-15	C	278.0	0.11290690478156595
SPY190215C00274000	2019-02-15	C	274.0	0.11812486916856693
SPY190315C00276000	2019-03-15	C	276.0	0.12283753251175747
SPY190418C00260000	2019-04-18	C	260.0	0.1486221542748649
SPY190315C00268000	2019-03-15	C	268.0	0.14659765736221353
SPY190315C00264000	2019-03-15	C	264.0	0.1505485763940055

Option Name	Expiration Date	Type	Strike	Implied Volatility
SPY190418C00272000	2019-04-18	C	272.0	0.12772451581247626
SPY190418P00262000	2019-04-18	P	262.0	0.11752034697081427
SPY190315P00274000	2019-03-15	P	274.0	0.16528594219471182
SPY190315P00278000	2019-03-15	P	278.0	0.23596284334616893
SPY190418P00270000	2019-04-18	P	270.0	0.13355500252960284
SPY190315P00266000	2019-03-15	P	266.0	0.11004668672371398
SPY190418C00281000	2019-04-18	C	281.0	0.11447732101011154
SPY190418P00258000	2019-04-18	P	258.0	0.12175334384069418
SPY190215P00264000	2019-02-15	P	264.0	0.11333376550308578
SPY190215P00268000	2019-02-15	P	268.0	0.09244671258170281
SPY190215P00276000	2019-02-15	P	276.0	0.24198950404096442
SPY190315P00264000	2019-03-15	P	264.0	0.11094075029768298
SPY190418P00272000	2019-04-18	P	272.0	0.14613784487595033
SPY190315P00276000	2019-03-15	P	276.0	0.20004307827376344
SPY190315P00268000	2019-03-15	P	268.0	0.11290280715278957
SPY190418P00260000	2019-04-18	P	260.0	0.11871556186919932
SPY190215P00274000	2019-02-15	P	274.0	0.16080216068745878
SPY190418P00256000	2019-04-18	P	256.0	0.12425845846190782
SPY190215P00266000	2019-02-15	P	266.0	0.10189264936520316
SPY190418C00283000	2019-04-18	C	283.0	0.112595460603914
SPY190215P00278000	2019-02-15	P	278.0	0.32880091606198675
SPY190215C00268000	2019-02-15	C	268.0	0.13919694924060208
SPY190215C00276000	2019-02-15	C	276.0	0.11406975329074713
SPY190418P00281000	2019-04-18	P	281.0	0.24426300507372298
SPY190215C00264000	2019-02-15	C	264.0	0.16773176974937565
SPY190418C00270000	2019-04-18	C	270.0	0.1311254013529824
SPY190315C00278000	2019-03-15	C	278.0	0.11911110499935687
SPY190315C00266000	2019-03-15	C	266.0	0.14605613620689764
SPY190418C00262000	2019-04-18	C	262.0	0.14051489817821766
SPY190315C00274000	2019-03-15	C	274.0	0.12726097155714888

A.2 AMZN Option Chain

Option Name	Expiration Date	Type	Strike	Implied Volatility
AMZN190315C01680000	2019-03-15	C	1680.0	0.27195602426748444
AMZN190215C01560000	2019-02-15	C	1560.0	0.2852750468898464
AMZN190215P01710000	2019-02-15	P	1710.0	0.7601725658797243
AMZN190215P01610000	2019-02-15	P	1610.0	0.1883249331618209
AMZN190418P01720000	2019-04-18	P	1720.0	0.428796570624232
AMZN190418P01620000	2019-04-18	P	1620.0	0.24735444646967036
AMZN190315P01655000	2019-03-15	P	1655.0	0.2981777752147001
AMZN190215C01690000	2019-02-15	C	1690.0	0.26820091335364926
AMZN190418P01675000	2019-04-18	P	1675.0	0.31055361413589827
AMZN190315P01600000	2019-03-15	P	1600.0	0.2071735133295474
AMZN190315P01700000	2019-03-15	P	1700.0	0.42151413305336255

Option Name	Expiration Date	Type	Strike	Implied Volatility
AMZN190315C01570000	2019-03-15	C	1570.0	0.2674809843301773
AMZN190215P01645000	2019-02-15	P	1645.0	0.2802935768576229
AMZN190215C01655000	2019-02-15	C	1655.0	0.26853240664352845
AMZN190418C01665000	2019-04-18	C	1665.0	0.27539276405978386
AMZN190315C01610000	2019-03-15	C	1610.0	0.277194306063835
AMZN190315C01710000	2019-03-15	C	1710.0	0.2662659545078912
AMZN190315P01560000	2019-03-15	P	1560.0	0.18919677685593705
AMZN190215P01680000	2019-02-15	P	1680.0	0.4927836537666028
AMZN190315C01645000	2019-03-15	C	1645.0	0.27459401913616055
AMZN190215P01570000	2019-02-15	P	1570.0	0.19241865943459904
AMZN190215C01700000	2019-02-15	C	1700.0	0.2682603045802592
AMZN190215C01600000	2019-02-15	C	1600.0	0.28551710231224897
AMZN190315P01690000	2019-03-15	P	1690.0	0.3727492042209791
AMZN190215C01645000	2019-02-15	C	1645.0	0.2744878222570395
AMZN190215P01690000	2019-02-15	P	1690.0	0.513898808023204
AMZN190315C01700000	2019-03-15	C	1700.0	0.2653441831583867
AMZN190315P01570000	2019-03-15	P	1570.0	0.19375597424519336
AMZN190315C01600000	2019-03-15	C	1600.0	0.2816049888005952
AMZN190418C01675000	2019-04-18	C	1675.0	0.2774616275601985
AMZN190315C01655000	2019-03-15	C	1655.0	0.2723185424609562
AMZN190418C01620000	2019-04-18	C	1620.0	0.2774217488515712
AMZN190418C01720000	2019-04-18	C	1720.0	0.27736047954510545
AMZN190315P01680000	2019-03-15	P	1680.0	0.3661725344255452
AMZN190418C01585000	2019-04-18	C	1585.0	0.276497440874729
AMZN190215C01610000	2019-02-15	C	1610.0	0.31359220099875995
AMZN190215P01560000	2019-02-15	P	1560.0	0.1983834166660943
AMZN190215C01710000	2019-02-15	C	1710.0	0.2704009253655553
AMZN190215P01600000	2019-02-15	P	1600.0	0.18337636347621908
AMZN190215C01570000	2019-02-15	C	1570.0	0.28880543825103017
AMZN190215P01700000	2019-02-15	P	1700.0	0.6548640429211394
AMZN190315C01690000	2019-03-15	C	1690.0	0.27180729009916105
AMZN190418P01595000	2019-04-18	P	1595.0	0.2216804362928776
AMZN190315P01645000	2019-03-15	P	1645.0	0.27630729138698723
AMZN190315C01560000	2019-03-15	C	1560.0	0.30713655759611397
AMZN190315P01610000	2019-03-15	P	1610.0	0.1861878002391142
AMZN190418P01665000	2019-04-18	P	1665.0	0.29625671903800477
AMZN190215C01680000	2019-02-15	C	1680.0	0.2670118936797237
AMZN190215P01655000	2019-02-15	P	1655.0	0.3282023451822188
AMZN190418C01655000	2019-04-18	C	1655.0	0.281185859914326
AMZN190315C01620000	2019-03-15	C	1620.0	0.2871799590947378
AMZN190315C01720000	2019-03-15	C	1720.0	0.2651385456094961
AMZN190215C01665000	2019-02-15	C	1665.0	0.26807598445726477
AMZN190418P01680000	2019-04-18	P	1680.0	0.343132006847645
AMZN190315C01585000	2019-03-15	C	1585.0	0.2819899463897471
AMZN190215C01630000	2019-02-15	C	1630.0	0.2751511015245677
AMZN190418C01700000	2019-04-18	C	1700.0	0.27353653822408613

Option Name	Expiration Date	Type	Strike	Implied Volatility
AMZN190418C01600000	2019-04-18	C	1600.0	0.2862739441035044
AMZN190315C01675000	2019-03-15	C	1675.0	0.269605034147687
AMZN190215P01585000	2019-02-15	P	1585.0	0.18601443151683758
AMZN190315P01665000	2019-03-15	P	1665.0	0.31851874592968876
AMZN190215P01720000	2019-02-15	P	1720.0	0.8250626639636887
AMZN190215P01620000	2019-02-15	P	1620.0	0.19983961149249846
AMZN190315P01595000	2019-03-15	P	1595.0	0.20540667921685807
AMZN190215P01675000	2019-02-15	P	1675.0	0.45730353011499586
AMZN190418P01645000	2019-04-18	P	1645.0	0.28212924137749634
AMZN190315P01630000	2019-03-15	P	1630.0	0.24576168840803453
AMZN190315P01675000	2019-03-15	P	1675.0	0.3389485473827938
AMZN190418P01600000	2019-04-18	P	1600.0	0.22347612454153387
AMZN190418P01700000	2019-04-18	P	1700.0	0.3772860597771452
AMZN190215P01595000	2019-02-15	P	1595.0	0.18335650948917165
AMZN190215P01630000	2019-02-15	P	1630.0	0.22236998428774002
AMZN190215P01665000	2019-02-15	P	1665.0	0.4026970168208832
AMZN190315P01585000	2019-03-15	P	1585.0	0.19265476090219014
AMZN190418C01680000	2019-04-18	C	1680.0	0.2812185433819471
AMZN190315P01620000	2019-03-15	P	1620.0	0.2304673621721585
AMZN190418P01655000	2019-04-18	P	1655.0	0.2938338008987934
AMZN190315C01630000	2019-03-15	C	1630.0	0.2802031851180679
AMZN190418C01645000	2019-04-18	C	1645.0	0.2810819313654204
AMZN190315C01595000	2019-03-15	C	1595.0	0.27487124323540024
AMZN190215C01675000	2019-02-15	C	1675.0	0.26953486225489154
AMZN190215C01620000	2019-02-15	C	1620.0	0.2825334187968613
AMZN190215C01720000	2019-02-15	C	1720.0	0.27154499307617813
AMZN190215C01585000	2019-02-15	C	1585.0	0.33621239860302193
AMZN190315C01665000	2019-03-15	C	1665.0	0.2735249038852389
AMZN190418P01560000	2019-04-18	P	1560.0	0.20630676728075423
AMZN190215C01670000	2019-02-15	C	1670.0	0.26815718092272045
AMZN190315C01590000	2019-03-15	C	1590.0	0.28512797392237826
AMZN190315C01635000	2019-03-15	C	1635.0	0.27782526772345423
AMZN190418C01640000	2019-04-18	C	1640.0	0.28482543233105595
AMZN190315C01660000	2019-03-15	C	1660.0	0.2713887953697263
AMZN190418P01565000	2019-04-18	P	1565.0	0.20039896221112108
AMZN190215C01580000	2019-02-15	C	1580.0	0.2855514121179136
AMZN190215C01725000	2019-02-15	C	1725.0	0.2732527530406747
AMZN190215P01555000	2019-02-15	P	1555.0	0.2039353377983698
AMZN190215C01625000	2019-02-15	C	1625.0	0.27873999017583745
AMZN190215P01635000	2019-02-15	P	1635.0	0.23765113957397774
AMZN190215P01590000	2019-02-15	P	1590.0	0.18603491966071947
AMZN190315P01670000	2019-03-15	P	1670.0	0.33132720176521163
AMZN190418C01575000	2019-04-18	C	1575.0	0.28621138209272223
AMZN190418P01605000	2019-04-18	P	1605.0	0.23465326070175757
AMZN190315P01625000	2019-03-15	P	1625.0	0.23554615352464758
AMZN190315C01555000	2019-03-15	C	1555.0	0.27919923146565756

Option Name	Expiration Date	Type	Strike	Implied Volatility
AMZN190315P01725000	2019-03-15	P	1725.0	0.5177042368427872
AMZN190315P01580000	2019-03-15	P	1580.0	0.19399605138832346
AMZN190418C01685000	2019-04-18	C	1685.0	0.2799948889886022
AMZN190215P01660000	2019-02-15	P	1660.0	0.36156299473989345
AMZN190215P01625000	2019-02-15	P	1625.0	0.20896817717100957
AMZN190215P01725000	2019-02-15	P	1725.0	0.8546877395161583
AMZN190418P01615000	2019-04-18	P	1615.0	0.24473523239955267
AMZN190418C01565000	2019-04-18	C	1565.0	0.285433442391398
AMZN190315P01660000	2019-03-15	P	1660.0	0.30779644656364263
AMZN190215P01580000	2019-02-15	P	1580.0	0.18820983369637023
AMZN190418P01640000	2019-04-18	P	1640.0	0.26743829097894145
AMZN190315P01635000	2019-03-15	P	1635.0	0.25762883598542274
AMZN190215P01670000	2019-02-15	P	1670.0	0.42779945656466667
AMZN190315P01590000	2019-03-15	P	1590.0	0.20057133091685106
AMZN190418P01685000	2019-04-18	P	1685.0	0.3128225053362834
AMZN190215C01660000	2019-02-15	C	1660.0	0.2693468347534804
AMZN190315P01555000	2019-03-15	P	1555.0	0.18802344036834015
AMZN190315C01725000	2019-03-15	C	1725.0	0.2645186085225371
AMZN190315C01625000	2019-03-15	C	1625.0	0.2755928405410493
AMZN190215C01590000	2019-02-15	C	1590.0	0.3245344796144139
AMZN190418C01605000	2019-04-18	C	1605.0	0.3455523883595186
AMZN190418P01575000	2019-04-18	P	1575.0	0.20974699493564303
AMZN190315C01670000	2019-03-15	C	1670.0	0.2732999489435454
AMZN190215C01635000	2019-02-15	C	1635.0	0.2770198885437168
AMZN190315P01640000	2019-03-15	P	1640.0	0.26502051926634806
AMZN190418P01635000	2019-04-18	P	1635.0	0.26573601891012755
AMZN190315C01695000	2019-03-15	C	1695.0	0.27083115199642716
AMZN190215C01575000	2019-02-15	C	1575.0	0.45143418434338695
AMZN190215P01705000	2019-02-15	P	1705.0	0.6873675075638325
AMZN190215P01605000	2019-02-15	P	1605.0	0.18405035023799027
AMZN190215P01650000	2019-02-15	P	1650.0	0.3069831648141222
AMZN190215C01685000	2019-02-15	C	1685.0	0.2687679471262276
AMZN190315P01615000	2019-03-15	P	1615.0	0.21666023127563164
AMZN190315P01715000	2019-03-15	P	1715.0	0.4832153734953507
AMZN190418P01660000	2019-04-18	P	1660.0	0.2945779351627125
AMZN190315C01605000	2019-03-15	C	1605.0	0.28084661039854864
AMZN190315C01705000	2019-03-15	C	1705.0	0.2667290841222114
AMZN190315P01575000	2019-03-15	P	1575.0	0.19115927274269826
AMZN190215P01695000	2019-02-15	P	1695.0	0.4995315946886302
AMZN190215C01640000	2019-02-15	C	1640.0	0.2743386978383564
AMZN190215P01565000	2019-02-15	P	1565.0	0.19599330394774142
AMZN190215C01715000	2019-02-15	C	1715.0	0.27242951990698305
AMZN190215C01615000	2019-02-15	C	1615.0	0.2718814039883548
AMZN190315P01685000	2019-03-15	P	1685.0	0.3667534464765388
AMZN190315C01650000	2019-03-15	C	1650.0	0.2719840003401422
AMZN190418P01555000	2019-04-18	P	1555.0	0.20284449048054493

Option Name	Expiration Date	Type	Strike	Implied Volatility
AMZN190418C01725000	2019-04-18	C	1725.0	0.2677525644717009
AMZN190418C01625000	2019-04-18	C	1625.0	0.2814957186998919
AMZN190215P01685000	2019-02-15	P	1685.0	0.5071174763047787
AMZN190418C01660000	2019-04-18	C	1660.0	0.2811121269869987
AMZN190315C01715000	2019-03-15	C	1715.0	0.26260445489907813
AMZN190315P01565000	2019-03-15	P	1565.0	0.18985415358677546
AMZN190315C01615000	2019-03-15	C	1615.0	0.33755210964271176
AMZN190215C01650000	2019-02-15	C	1650.0	0.27077783404103933
AMZN190315P01695000	2019-03-15	P	1695.0	0.4090758540746196
AMZN190215C01605000	2019-02-15	C	1605.0	0.3114741171717339
AMZN190215P01575000	2019-02-15	P	1575.0	0.19041726046510973
AMZN190215C01705000	2019-02-15	C	1705.0	0.2690154146355436
AMZN190418C01635000	2019-04-18	C	1635.0	0.280478848215869
AMZN190315C01640000	2019-03-15	C	1640.0	0.2775746050393185
AMZN190418P01625000	2019-04-18	P	1625.0	0.24744296012936956
AMZN190418C01555000	2019-04-18	C	1555.0	0.28673592735739317
AMZN190418P01725000	2019-04-18	P	1725.0	0.4001241937622695
AMZN190315P01650000	2019-03-15	P	1650.0	0.2869187108695964
AMZN190215P01615000	2019-02-15	P	1615.0	0.19189350440374117
AMZN190215C01565000	2019-02-15	C	1565.0	0.29585641811491775
AMZN190215P01715000	2019-02-15	P	1715.0	0.7592116231503694
AMZN190315C01685000	2019-03-15	C	1685.0	0.27697383900127753
AMZN190215P01640000	2019-02-15	P	1640.0	0.2588814420773245
AMZN190315P01705000	2019-03-15	P	1705.0	0.4525746469912322
AMZN190315P01605000	2019-03-15	P	1605.0	0.19807775307189474
AMZN190215C01695000	2019-02-15	C	1695.0	0.2687012387053741

B Solution Source Code

B.1 Question 1 Implementation

B.1.1 Bloomberg Terminal 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.
```

```

  # Empty vector to store names
  names <- c()

  # Iterate over each date and price
  for (date in dates) {
    for (price in prices) {
      # Building option name
      name <- paste(security, date, paste(type, price, sep = ""), suffix)

      # Appending to list of option names
      names <- c(names, name)
    }
  }

  # Returning names
  names
}


#---------------------------------
# DATA1
#---------------------------------


# Define Start and End times (DATA1)
data1Start <- ISOdatetime(year = 2019, month = 2, day = 6,
                          hour = 9, min = 30, sec = 0)
data1End <- ISOdatetime(year = 2019, month = 2, day = 6,
                        hour = 16, min = 0, sec = 0)

# Defining time zone
timeZone = "America/New_York"

# Defining top-level securities
securities <- c("SPY US Equity", "AMZN US Equity", "VIX Index")

# Getting prices for each of the top-level securities
for (security in securities) {
  data <- getPrice(security, data1Start, data1End, timeZone)
  write.csv(data, file = paste(security, "DATA1", "csv", sep = "."),
            row.names = FALSE)
}

# Expiration dates
expDates <- c("2/15/19", "3/15/19", "4/18/19")

# Defining put and call prices for SPY and AMZN options
# Grabbing prices for 15% +/- current price

# Defining bounds
lowerBoundPct <- 0.85
upperBoundPct <- 1.15

# Current SPY price
spyCurrent <- 270
spyPrices <- c(floor(
  lowerBoundPct * spyCurrent):ceiling(upperBoundPct * spyCurrent))

# Function to round to the nearest 'base', given an input 'x'. This is to
# compute strike prices for AMZN options, which are in intervals of 5.
```

```
# Source: http://r.789695.n4.nabble.com/Rounding-to-the-nearest-5-td863189.html
mround <- function(x, base) {
  base * round(x / base)
}

# Current AMZN price (need to do this manually because of option strikes)
amznCurrent <- 1640
roundingLevel <- 5
amznPrices <- seq(mround(amznCurrent * lowerBoundPct, roundingLevel),
                  mround(amznCurrent * upperBoundPct, roundingLevel), by=5)

# Creating option names for SPY and AMZN
spyOptions <- createOptionName("SPY", expDates, spyPrices, "C", "Equity")
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

B.2 Question 2 Implementation

B.2.1 Optimization Method Convergence Comparison

```
from context import fe621

from datetime import datetime
import numpy as np
import pandas as pd


# Defining dates
data1_date = '2019-02-06'

# Loading DATA1
spy_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/SPY',
                                date=data1_date)
amzn_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/AMZN',
                                 date=data1_date)

# Loading Risk-free rate (effective federal funds rate)
rf = pd.read_csv('Homework 1/data/ffr.csv')

# Setting comparison tolerance level
tol = 1e-3

# Number of input options
input_count = len(spy_data1.columns) - 1

def compareConvergenceTime():
    """Function to compare the convergence times of the Newton and Bisection
    method solvers, on the SPY option chain.
    """

    # Newton's Method
    start = datetime.now().timestamp()
    spy_vol_newton = fe621.util.computeAvgImpliedVolNewton(
        data=spy_data1,
        name='SPY',
        rf=rf[data1_date][0],
        current_date=data1_date,
        tol=tol
    )
    end = datetime.now().timestamp()

    # Computing time  and number of options for Newton
    newton_time = end - start
    newton_count = spy_vol_newton.count(axis=0)[0]

    # Bisection Method
    start = datetime.now().timestamp()
    spy_vol_bisection = fe621.util.computeAvgImpliedVolNewton(
        data=spy_data1,
        name='SPY',
        rf=rf[data1_date][0],
        current_date=data1_date,
        tol=tol
    )
    end = datetime.now().timestamp()

    # Computing time and number of options for Bisection
```

```
    bisection_time = end - start
    bisection_count = spy_vol_bisection.count(axis=0)[0]

    # Building DataFrame, and saving to CSV
    convergence_table = pd.DataFrame({
        'Number of Input Options': [input_count, input_count],
        'Number of Output Options': [newton_count, bisection_count],
        'Number of Dropped Options': [input_count - newton_count,
                                      input_count - bisection_count],
        'Time Elapsed for Computation (s)': [newton_time, bisection_time],
        'Average Time per Option (s)': [newton_time / input_count,
                                        bisection_time / input_count]
    })
    convergence_table = convergence_table.T  # Transposing so cols are methods
    convergence_table.columns = ['Newton Method', 'Bisection Method']
    convergence_table.to_csv('Homework 1/bin/imp_vol_convergence.csv')

if __name__ == '__main__':
    compareConvergenceTime()
```

question_solutions/question_2_convergence.py

B.2.2 Implied Volatility Computation

```
from context import fe621

import numpy as np
import pandas as pd


# Defining dates
data1_date = '2019-02-06'
data2_date = '2019-02-07'

# Loading DATA1
spy_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/SPY',
                                date=data1_date)
amzn_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/AMZN',
                                 date=data1_date)
vix_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/VIX',
                                date=data1_date)

# Loading DATA2
spy_data2 = fe621.util.loadData(folder_path='Homework 1/data/DATA2/SPY',
                                date=data2_date)
amzn_data2 = fe621.util.loadData(folder_path='Homework 1/data/DATA2/AMZN',
                                 date=data2_date)
vix_data2 = fe621.util.loadData(folder_path='Homework 1/data/DATA2/VIX',
                                date=data2_date)

# Loading Risk-free rate (effective federal funds rate)
rf = pd.read_csv('Homework 1/data/ffr.csv')

# Tolerance level for optimization
tol = 1e-6

def computeImpVolatilities():
    """Function to compute the implied volatilities for the SPY and AMZN option
    chains, for all maturities. Computed implied volatilities are output to
```

```
    CSV files.
    """

    # SP 500
    spy_data1_vol = fe621.util.computeAvgImpliedVolBisection(
                                                            data=spy_data1,
                                                            name='SPY',
                                                            rf=rf[data1_date][0],
                                                            current_date=data1_date,
                                                            tol=tol)
    # Saving to CSV
    spy_data1_vol.to_csv('Homework 1/bin/spy_data1_vol.csv', index=False)

    # AMZN
    amzn_data1_vol = fe621.util.computeAvgImpliedVolBisection(
                                                             data=amzn_data1,
                                                             name='AMZN',
                                                             rf=rf[data1_date][0],
                                                             current_date=data1_date,
                                                             tol=tol)
    # Saving to CSV
    amzn_data1_vol.to_csv('Homework 1/bin/amzn_data1_vol.csv', index=False)


if __name__ == "__main__":
    # Part 1 - Implied Volatility Computation
    computeImpVolatilities()
```

question_solutions/question_2_imp_vol.py

B.2.3 Volatility Plots

```
from context import fe621

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


# Loading implied volatility data from CSV files
spy_imp_vol = pd.read_csv('Homework 1/bin/spy_data1_vol.csv',
                          index_col=False, header=0)
amzn_imp_vol = pd.read_csv('Homework 1/bin/amzn_data1_vol.csv',
                           index_col=False, header=0)

# Defining date of DATA1
data1_date = '2019-02-06'


def plot2DVolSmile(data: pd.DataFrame, name: str, save_loc: str):
    """Function to plot a 2D Volatility Smile for a given option chain.

    Arguments:
        data {pd.DataFrame} -- Input data containing implied volatilities.
        name {str} -- Name of the underlying asset.
        save_loc {str} -- Location (folder) to save the output image.
    """

```

```
    # Iterating through types of options for 2 separate put/call imp vol plots
    for option_type_group in data.groupby('type'):
        # Isolating current option type
        option_type = option_type_group[0]

        # Iterating through expiration dates for individual lines for each
        for exp_date_group in option_type_group[1].groupby('expiration'):
            # Isolating current expiration date
            exp_date = exp_date_group[0]

            # Sorting data to be ascending on 'strike'
            plt_data = exp_date_group[1].sort_values(by='strike')

            # Plotting strike vs implied vol
            plt.plot(plt_data['strike'], plt_data['implied_vol'],
                     label=('Maturity on ' + exp_date))

        # Formatting plot
        #================

        ax = plt.gca()  # Get current axes

        # Setting y ticks and label
        ax.set_yticklabels(['{:,.1%}'.format(i) for i in ax.get_yticks()])
        ax.set_ylabel('Implied Volatility')
        # Setting x ticks and label
        ax.set_xticklabels(['$%i' % i for i in ax.get_xticks()])
        ax.set_xlabel('Strike Price')

        # Setting legend and setting plot dimensions to tight
        plt.legend()
        plt.tight_layout()

        # Saving to file
        full_option_type = 'Call' if (option_type == 'C') else 'Put'
        fname = '_'.join([name, full_option_type, '2DVolSmile.png'])
        plt.savefig(fname=(save_loc + '/' + fname))

        # Closing plot for next one
        plt.close()


def plot3DVolatilitySurface(data: pd.DataFrame, name: str, save_loc: str):
    """Fuction to plot a 3D Volatility Surface for a given option chain.

    Arguments:
        data {pd.DataFrame} -- Input data containing implied volatilities
        name {str} -- Name of the underlying asset.
        save_loc {str} -- Location (folder) to save the output image.
    """

    # Iterating through types of options for 2 separate put/call imp vol plots
    for option_type_group in data.groupby('type'):
        # Isolating current option type
        option_type = option_type_group[0]

        # Isolating plot data
        plot_data = option_type_group[1]

        # Creating new column with time to maturity information for each option
        ttm = plot_data.apply(lambda row: fe621.util.getTTM(
```

```
                                   name=row.loc['name'],
                                   current_date=data1_date),
                                 axis=1)
        # Converting TTM to days
        ttm_days = ttm * 365

        # Isolating data for each axis
        x = np.array(ttm_days)
        y = np.array(plot_data['strike'])
        z = np.array(plot_data['implied_vol'])

        # Plotting surface
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot_trisurf(x, y, z, cmap='plasma')

        # Formatting plot
        #----------------

        # Setting x label
        ax.set_xlabel('TTM (Days)')
        # Setting y label
        ax.set_ylabel('Strike Price ($)')
        # Setting z label
        ax.set_zlabel('Implied Volatility')

        # Modifying z ticks to be percentages
        ax.set_zticklabels(['{:,.0%}'.format(i) for i in ax.get_zticks()])

        # Setting plot dimensions to tight
        plt.tight_layout()

        # Saving to file
        full_option_type = 'Call' if (option_type == 'C') else 'Put'
        fname = '_'.join([name, full_option_type, '3DVolSurface.png'])
        plt.savefig(fname=(save_loc + '/' + fname))

        # Closing plot for next one
        plt.close()

if __name__ == '__main__':
    # Plotting 2D Volatility Smile for AMZN and SPY option chains
    plot2DVolSmile(data=amzn_imp_vol, name='AMZN',
                   save_loc='Homework 1/bin/vol_smile/')
    plot2DVolSmile(data=spy_imp_vol, name='SPY',
                   save_loc='Homework 1/bin/vol_smile/')

    # Plotting 3D Volatility Surface for AMZN and SPY option chains
    plot3DVolatilitySurface(data=spy_imp_vol, name='SPY',
                            save_loc='Homework 1/bin/vol_surface/')
    plot3DVolatilitySurface(data=amzn_imp_vol, name='AMZN',
                            save_loc='Homework 1/bin/vol_surface/')
```

question_solutions/question_2_vol_plots.py

B.3 Question 3 Implementation

B.3.1 Truncation Error Analysis

```
from context import fe621

import numpy as np
import pandas as pd


def truncationErrorAnalysis():
    """Function to analyze the truncation error of the Trapezoidal and Simpson's
    quadature rules.
    """

    # Objective function
    def f(x: float) -> float:
        return np.where(x == 0.0, 1.0, np.sin(x) / x)

    # Setting values for N
    N = np.power(10, np.arange(3, 8))

    # Setting values for a
    a = np.power(10, np.arange(2, 7))

    trapezoidal_vals = np.ndarray((N.size, a.size))
    simpsons_vals = np.ndarray((N.size, a.size))

    # Building function approximation table, varying N and A
    for i in range(0, N.size):
        for j in range(0, a.size):
            # Trapezoidal rule approximation
            trapezoidal_vals[i, j] = fe621.numerical_integration \
                .trapezoidalRule(f=f, N=N[i], start=-a[j], stop=a[j])
            # Simpsons rule trunc approximation
            simpsons_vals[i, j] = fe621.numerical_integration \
                .simpsonsRule(f=f, N=N[i], start=-a[j], stop=a[j])

    # Computing the absolute difference from Pi (i.e. trunc error)
    # and casting to DataFrame
    trapezoidal_df = pd.DataFrame(np.abs(trapezoidal_vals - np.pi))
    simpsons_df = pd.DataFrame(np.abs(simpsons_vals - np.pi))

    # Setting row and column names
    trapezoidal_df.columns = ['N = ' + str(i) for i in N]
    trapezoidal_df.index = ['a = ' + str(i) for i in a]
    simpsons_df.columns = ['N = ' + str(i) for i in N]
    simpsons_df.index = ['a = ' + str(i) for i in a]

    # Saving to CSV
    trapezoidal_df.to_csv(
        'Homework 1/bin/numerical_integration/trapezoidal_trunc_error.csv',
        header=True, index=True, float_format='%.8e'
    )
    simpsons_df.to_csv(
        'Homework 1/bin/numerical_integration/simpsons_trunc_error.csv',
        header=True, index=True, float_format='%.8e'
    )


if __name__ == '__main__':
```

```python
    # Part 2 - Truncation Error Analysis
    truncationErrorAnalysis()
```

question_solutions/question_3_trunc_error.py

B.3.2 Convergence Segment Analysis

```python
from context import fe621

import numpy as np
import pandas as pd


def convergenceSegmentLimit():
    """Function to compute the number of segments required for convergence of
    various quadrature methods.
    """

    # Objective function
    def f(x: float) -> float:
        return np.where(x == 0.0, 1.0, np.sin(x) / x)

    # Setting target tolerance level for termination
    epsilon = 1e-3

    # Using Trapezoidal rule
    trapezoidal_result = fe621.numerical_integration.convergenceApproximation(
        f=f,
        rule=fe621.numerical_integration.trapezoidalRule,
        epsilon=epsilon
    )

    # Using Simpson's rule
    simpsons_result = fe621.numerical_integration.convergenceApproximation(
        f=f,
        rule=fe621.numerical_integration.simpsonsRule,
        epsilon=epsilon
    )

    # Building DataFrame of results for output
    results = pd.DataFrame(np.abs(np.array([trapezoidal_result,
                                            simpsons_result])))

    # Setting row and column names
    results.columns = ['Estimated Area', 'Segments']
    results.index = ['Trapezoidal Rule', 'Simpson\'s Rule']

    # Saving to CSV
    results.to_csv('Homework 1/bin/numerical_integration/convergence.csv',
                   header=True, index=True, float_format='%.8e')


if __name__ == '__main__':
    # Part 3 - Convergence Analysis
    convergenceSegmentLimit()
```

question_solutions/question_3_convergence.py

B.3.3 Arbitrary Function Convergence Segment Analysis

```python
from context import fe621

import numpy as np
import pandas as pd


def arbitraryFunctionSegmentAnalysis():
    """Function to analyze number of segments required for an arbitrary function
    to converge under the Trapezoidal and Simpson's quadrature rules.
    """

    # Defining objective function
    def g(x: float) -> float:
        return 1 + np.exp(-1 * np.power(x, 2)) * np.cos(8 * np.power(x, 2/3))

    # Setting target tolerance level for termination
    epsilon = 1e-4

    # Setting start and stop limits
    start = 0
    stop = 2

    # Trapezoidal rule
    trapezoidal_result = fe621.numerical_integration.convergenceApproximation(
        f=g,
        rule=fe621.numerical_integration.trapezoidalRule,
        start=start,
        stop=stop,
        epsilon=epsilon
    )

    # Simpson's rule
    simpsons_result = fe621.numerical_integration.convergenceApproximation(
        f=g,
        rule=fe621.numerical_integration.simpsonsRule,
        start=start,
        stop=stop,
        epsilon=epsilon
    )

    # Building DataFrame of results for output
    results = pd.DataFrame(np.abs(np.array([trapezoidal_result,
                                            simpsons_result])))

    # Setting row and column names
    results.columns = ['Estimated Area', 'Segments']
    results.index = ['Trapezoidal Rule', 'Simpson\'s Rule']

    # Saving to CSV
    results.to_csv('Homework 1/bin/numerical_integration/arb_convergence.csv',
                   header=True, index=True, float_format='%.8e')


if __name__ == '__main__':
    # Part 4 - Arbitrary Function
    arbitraryFunctionSegmentAnalysis()
```

question_solutions/question_3_arbitrary_area.py