

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 model¹. 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 repository² 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 ETF³

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 Index⁴

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 *Rbblpapi*⁵, which provides an R interface to data on the Bloomberg Terminal⁶. 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 package⁸.

1.3 Option Naming Convention

A modern convention for naming option contracts was proposed by the Options Clearing Commission (OCC) in 2008⁹, 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 Symbolology 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** - $\frac{1}{1000}$ 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 fe621¹⁰ package created for this class.

2.1 Black-Scholes Model Formulas

With the probabilities d_1 and d_2 defined as:

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

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

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

```

1 from typing import Tuple
2
3 import numpy as np
4
5
6 def computeD1D2(current: float, volatility: float, ttm: float, strike: float,
7                 rf: float) -> Tuple[float, float]:
8     """Helper function to compute the risk-adjusted priors of exercising the
9     option contract, and keeping the underlying asset. This is used in the
10    computation of both the Call and Put options in the
11    Black-Scholes-Merton framework.
12
13    Arguments:
14        current {float} -- Current price of the underlying asset.
15        volatility {float} -- Volatility of the underlying asset price.
16        ttm {float} -- Time to expiration (in years).
17        strike {float} -- Strike price of the option contract.
18        rf {float} -- Risk-free rate (annual).
19
20    Returns:
21        Tuple[float, float] -- Tuple with d1, and d2 respectively.
22    """
23
24    d1 = (np.log(current / strike) + (rf + ((volatility ** 2) / 2)) * ttm) \
25          / (volatility * np.sqrt(ttm))
26    d2 = d1 - (volatility * np.sqrt(ttm))
27
28    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:

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

```

1 from .util import computeD1D2
2
3 from scipy.stats import norm
4 import numpy as np
5
6
7 def blackScholesPut(current: float, volatility: float, ttm: float,
8                     strike: float, rf: float) -> float:
9     """Function to compute the Black-Scholes-Merton price of a European Put
10    Option, parameterized by the current underlying asset price, volatility,
11    time to expiration, strike price, and risk-free rate.
12
13    Arguments:
14        current {float} -- Current price of the underlying asset.
15        volatility {float} -- Volatility of the underlying asset price.
16        ttm {float} -- Time to expiration (in years).
17        strike {float} -- Strike price of the option contract.
18        rf {float} -- Risk-free rate (annual).
19
20    Returns:
21        float -- Price of a European Put Option contract.
22    """
23
24    d1, d2 = computeD1D2(current, volatility, ttm, strike, rf)
25
26    put = (strike * np.exp(-1 * rf * ttm) * norm.cdf(-1 * d2)) \
27          - (strike * norm.cdf(-1 * d1))
28
29    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:

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

```

1 from .util import computeD1D2
2
3 from scipy.stats import norm
4 import numpy as np
5
6
7 def blackScholesCall(current: float, volatility: float, ttm: float,
8                      strike: float, rf: float) -> float:
9     """Function to compute the Black-Scholes-Merton price of a European Call

```

```

10 Option, parameterized by the current underlying asset price, volatility,
11 time to expiration, strike price, and risk-free rate.
12
13 Arguments:
14     current {float} -- Current price of the underlying asset.
15     volatility {float} -- Volatility of the underlying asset price.
16     ttm {float} -- Time to expiration (in years).
17     strike {float} -- Strike price of the option contract.
18     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 .

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

```

1 from .util import computeD1D2
2
3 from scipy.stats import norm
4
5 import numpy as np
6
7
8 def callDelta(current: float, volatility: float, ttm: float, strike: float,
9             rf: float) -> float:
10     """Function to compute the Delta of a call option using the Black-Scholes

```



```

11     formula.
12
13     Arguments:
14         current {float} -- Current price of the underlying asset.
15         volatility {float} -- Volatility of the underlying asset price.
16         ttm {float} -- Time to expiration (in years).
17         strike {float} -- Strike price of the option contract.
18         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 Option contract.
43     """
44
45     d1, d2 = computeD1D2(current, volatility, ttm, strike, rf)
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 midpoint = m ;
repeat
 $m = \frac{a+b}{2}$;
 if $f(a) \times f(m) < 0$ **then**
 $b = m$
 end
 if $f(b) \times f(m) < 0$ **then**
 $a = m$
 end
until $(b - a) < \epsilon$;
return $\frac{a+b}{2}$;

```

1 from typing import Callable
2 import numpy as np
3
4
5 def bisectionSolver(f: Callable, a: float, b: float,
6                   tol: float=10e-6) -> float:
7     """Bisection method solver, implemented using recursion.
8
9     Arguments:
10         f {Callable} -- Function to be optimized.
11         a {float} -- Lower bound.
12         b {float} -- Upper bound.
13
14     Keyword Arguments:
15         tol {float} -- Solution tolerance (default: {10e-6}).
16
17     Raises:
18         Exception -- Raised if no solution is found.
19
20     Returns:
21         float -- Solution to the function s.t. f(x) = 0.
22     """
23
24     # Compute midpoint
25     mid = (a + b) / 2
26
27     # Check if estimate is within tolerance
28     if (b - a) < tol:
29         return mid
30
31     # Evaluate function at midpoint
32     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 2.¹¹

Algorithm 2: Newton's Method

Input: A differentiable function $f : \mathbb{R}^a \rightarrow \mathbb{R}^b \forall a, b \in \mathbb{N}_{>0}$. 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

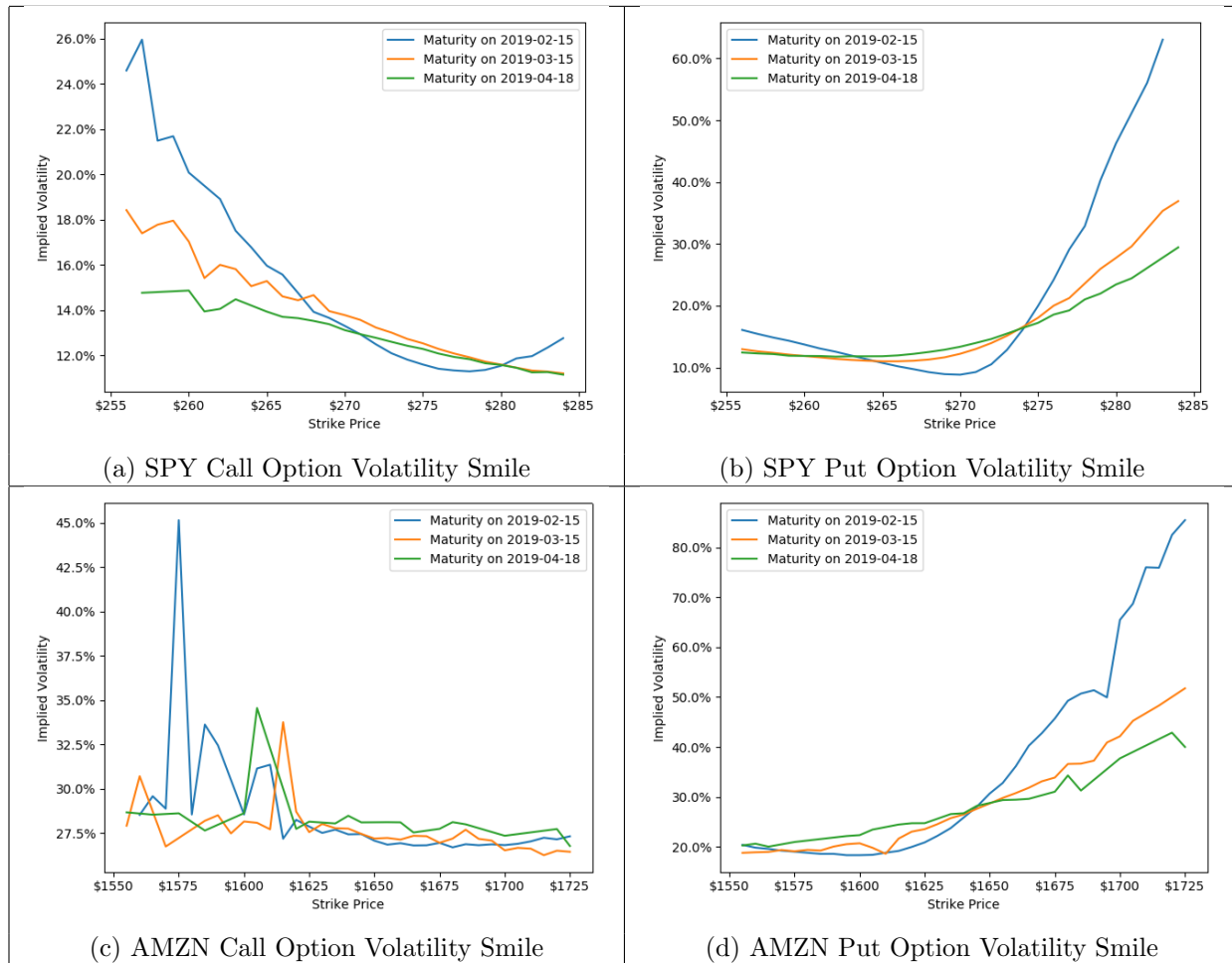


Figure 1: Volatility Smiles of call and put option chains on AMZN and SPY. Plots the relationship between the strike price and implied volatility for various maturities.

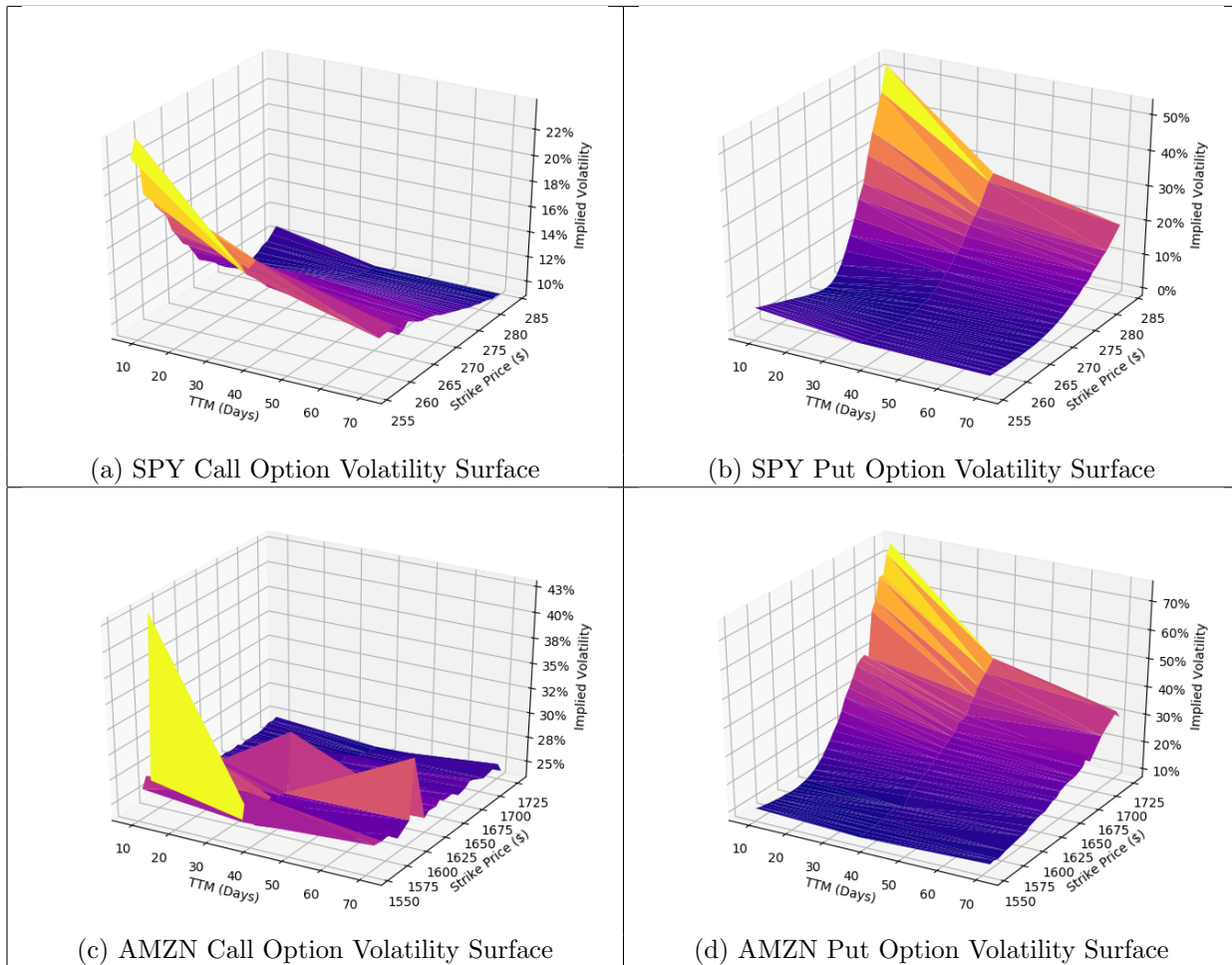


Figure 2: Volatility Surfaces of call and put option chains on AMZN and SPY. Plots the relationship between the strike price, time to maturity, and the implied volatility.

3 Numerical Integration

3.1 Quadrature Methods

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

Let data = \mathbf{x}

Let i^{th} element of $\mathbf{x} = x_i$

3.1.1 Trapezoidal Rule

Let Trapezoidal rule approximation = $T_N(f)$

$$\begin{aligned} \Rightarrow T_N(f) &= \sum_{i=1}^N \left[\left(\frac{f(x_{i-1}) + f(x_i)}{2} \right) \times h \right] \\ \Rightarrow h \times \sum_{i=1}^N \left[\frac{f(x_{i-1}) + f(x_i)}{2} \right] &= h \times \left(\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{N-1}) + \frac{1}{2}f(x_N) \right) \\ \therefore T_N(f) &= hf(\mathbf{x}) - \frac{h}{2}(f(x_0) + f(x_N)) \end{aligned}$$

```

1 from typing import Callable
2 import numpy as np
3
4
5 def trapezoidalRule(f: Callable, N: float, start: float=-1e6,
6                     stop: float=1e6) -> float:
7     """Function to approximate the numeric integral of a function, f, using
8     the Trapezoidal rule.
9
10    Arguments:
11        f {Callable} -- Function who's integral is to be estimated.
12        N {int} -- Number of nodes to consider.
13
14    Keyword Arguments:
15        start {float} -- Starting point (default: {-1e6}).
16        stop {float} -- Stopping point (default: {1e6}).
17
18    Returns:
19        float -- Approximation of the area under the function.
20    """
21
22    # Building values for approximation, and getting step size
23    x, h = np.linspace(start=start, stop=stop, num=N, retstep=True)
24
25    # Estimating area using trapezoidal rule, return
26    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).

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

Note that $\left(\frac{x_{i-1} + x_i}{2}\right)$ is the midpoint between the points in \mathbf{x} .

Let the above = \mathbf{x}_{mid}

$$\therefore S_N(f) \approx \frac{h}{6} (2f(\mathbf{x}) - (f(x_0) + f(x_N)) + 4f(\mathbf{x}_{\text{mid}}))$$

```

1 from typing import Callable
2 import numpy as np
3
4
5 def simpsonsRule(f: Callable, N: float, start: float=-1e6,
6                 stop: float=1e6) -> float:
7     """Function to approximate the numeric integral of a function, f, using
8     Simpson's rule.
9
10    Arguments:
11        f {Callable} -- Function for which the integral is to be estimated.
12        N {float} -- Number of nodes to consider.
13
14    Keyword Arguments:
15        start {float} -- Starting point (default: {-1e6}).
16        stop {float} -- Stopping point (default: {1e6}).
17
18    Returns:
19        float -- Approximation of the area under the function.
20    """
21
22    # Building values for approximation, and getting step size
23    x, h = np.linspace(start=start, stop=stop, num=N, retstep=True)
24
25    # Computing midpoints
26    x_mid = np.array([(x[i - 1] + x[i]) / 2 for i in range(1, N)])
27
28    # Estimating using Simpson's rule
29    area = np.sum(2 * f(x)) - (f(start) + f(stop)) + (4 * np.sum(f(x_mid)))
30
31    # Scaling area
32    area *= (h / 6)
33
34    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 .

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

It is known 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:

Truncation error for approximation with parameters a 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: Simpson's 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 explored 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 .

Let approximation with parameter $N = I_N$

Repeat while: $|I_N - I_{N_{old}}| > \epsilon$

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` package¹² sub-module used in this analysis is presented below.

```

1 from typing import Callable, Tuple
2 import numpy as np
3
4
5 def convergenceApproximation(f: Callable, rule: Callable, epsilon: float=1e-3,
6                             start: float=-1e6, stop: float=1e6) \
7     -> Tuple[float, int]:
8     """Function to approximate the numeric integral of a function, f, using
9     a given quadrature rule and a tolerance level epsilon.
10
11     Arguments:
12         f {Callable} -- Function for which the integral is to be estimated.
13         rule {Callable} -- Function to be used to approximate area. Must take
14             positional arguments f, N, start and stop.
15
16     Keyword Arguments:
17         epsilon {float} -- Tolerance level (default: {1e-3}).
18         start {float} -- Starting point (default: {-1e6}).
19         stop {float} -- Stopping point (default: {1e6}).
20
21     Returns:
22         Tuple[float, int] -- Approximation of the area under the function
23             and the number of segments (area, segments).
24
25     """
26
27     # Flags
28     area_old = 0
29     area_new = 1
30     N = 1
31
32     while (np.abs(area_new - area_old) > epsilon):
33         # Set new area to old area
34         area_old = area_new

```

12. Weerawarana 2019

```

34
35     # Increase N by powers of 10
36     N *= 10
37
38     # Computing area with given parameters
39     area_new = rule(f=f, N=N, start=start, stop=stop)
40
41     # Log
42     print('On iteration {0} method {1} convergence {2} val {3}'.format(
43         N, str(rule),
44         '{:.5e}'.format(np.abs(area_new - area_old)),
45         area_new))
46
47     # Return final area and number of segments
48     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:

$$g(x) = 1 + e^{-x^2} \cos(8x^{\frac{2}{3}})$$

$$\int_0^2 g(x) dx$$

| | 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

```

1 library("Rblpapi")
2
3 # Connect to Bloomberg Terminal backend service
4 blpConnect(host = "localhost", port = 8194)
5
6
7 #-----
8 # Data Download Functionality
9 #-----
10
11
12 getPrice <- function(security, startTime, endTime, timeZone) {
13   # Downloads and returns the closing price of a given security
14   # for each minute in the trading day.
15   #
16   # Args:
17   #   security: Name of the security to be downloaded.
18   #   startTime: Datetime object with the start time.
19   #   endTime: Datetime object with the end time.
20   #   timeZone: Time zone of the target start and end times.
21   #
22   # Returns:
23   #   DataFrame with the closing price for each minute in the
24   #   trading day.
25
26   # Getting price data
27   data <- getBars(security = security, barInterval = 1,
28                   startTime = startTime, endTime = endTime,
29                   tz = timeZone)
30
31   # Isolate time and closing price
32   data <- data[c("times", "close")]
33
34   # Rename columns
35   colnames(data) <- c("Dates", "Close")
36
37   # Return
38   data
39 }
40
41
42 createOptionName <- function(security, dates, prices, type, suffix) {
43   # Creates the Bloomberg-standard option name, given a security, date, price,
44   # option type and suffix.
45   #
46   # Args:
47   #   security: Name of the security to be included in the option price.
48   #   dates: Dates to be included in option name.
49   #   prices: Prices to be included in the option name.
50   #   type: Type of the option ("C" or "P").
51   #   suffix: Suffix for option name (typically "Index" or "Equity").
52   #
53   # Returns:
54   #   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 15% +/- current price
104
105 # Defining bounds
106 lowerBoundPct <- 0.85
107 upperBoundPct <- 1.15
108
109 # Current SPY price
110 spyCurrent <- 270
111 spyPrices <- c(floor(
112   lowerBoundPct * spyCurrent):ceiling(upperBoundPct * spyCurrent))
113
114 # Function to round to the nearest 'base', given an input 'x'. This is to
115 # compute strike prices for AMZN options, which are in intervals of 5.

```

```

116 # Source: http://r.789695.n4.nabble.com/Rounding-to-the-nearest-5-td863189.html
117 mround <- function(x, base) {
118   base * round(x / base)
119 }
120
121 # Current AMZN price (need to do this manually because of option strikes)
122 # Closest option price to 95% of price is at $1557.50 and 105% is $1722.50
123 amznCurrent <- 1640
124 roundingLevel <- 5
125 amznPrices <- seq(mround(amznCurrent * lowerBoundPct, roundingLevel),
126                  mround(amznCurrent * upperBoundPct, roundingLevel), by=5)
127
128 # Creating option names for SPY and AMZN
129 spyOptions <- createOptionName("SPY", expDates, spyPrices, "C", "Equity")
130 spyOptions <- c(spyOptions, createOptionName("SPY", expDates, spyPrices,
131                                             "P", "Equity"))
132
133 amznOptions <- createOptionName("AMZN", expDates, amznPrices, "C", "Equity")
134 amznOptions <- c(amznOptions, createOptionName("AMZN", expDates, amznPrices,
135                                             "P", "Equity"))
136
137 # Getting prices for each of the options
138 for (option in c(amznOptions, spyOptions)) {
139   data <- getPrice(option, data1Start, data1End, timeZone)
140   # Only print to file if option exists
141   if (all(dim(data) > 0)) {
142     optionFileName <- gsub("/", "-", option) # Need to do this for Windows
143     write.csv(data, file = paste(optionFileName, "csv", sep = "."),
144              row.names = FALSE)
145   }
146 }
147
148
149 #-----
150 # DATA2
151 #-----
152
153 # Define Start and End times (DATA2)
154 data2Start <- ISOdatetime(year = 2019, month = 2, day = 7,
155                           hour = 9, min = 30, sec = 0)
156 data2End <- ISOdatetime(year = 2019, month = 2, day = 7,
157                          hour = 16, min = 0, sec = 0)
158
159 # Getting prices for each of the top-level securities
160 for (security in securities) {
161   data <- getPrice(security, data2Start, data2End, timeZone)
162   write.csv(data, file = paste(security, "DATA2", "csv", sep = "."),
163            row.names = FALSE)
164 }

```

question_solutions/question_1.R

B.2 Question 2 Implementation

B.2.1 Optimization Method Convergence Comparison

```

1 from context import fe621
2
3 from datetime import datetime
4 import numpy as np
5 import pandas as pd
6
7
8 # Defining dates
9 data1_date = '2019-02-06'
10
11 # Loading DATA1
12 spy_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/SPY',
13                                date=data1_date)
14 amzn_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/AMZN',
15                                date=data1_date)
16
17 # Loading Risk-free rate (effective federal funds rate)
18 rf = pd.read_csv('Homework 1/data/ffr.csv')
19
20 # Setting comparison tolerance level
21 tol = 1e-3
22
23 # Number of input options
24 input_count = len(spy_data1.columns) - 1
25
26 def compareConvergenceTime():
27     """Function to compare the convergence times of the Newton and Bisection
28     method solvers, on the SPY option chain.
29     """
30
31     # Newton's Method
32     start = datetime.now().timestamp()
33     spy_vol_newton = fe621.util.computeAvgImpliedVolNewton(
34         data=spy_data1,
35         name='SPY',
36         rf=rf[data1_date][0],
37         current_date=data1_date,
38         tol=tol
39     )
40     end = datetime.now().timestamp()
41
42     # Computing time and number of options for Newton
43     newton_time = end - start
44     newton_count = spy_vol_newton.count(axis=0)[0]
45
46     # Bisection Method
47     start = datetime.now().timestamp()
48     spy_vol_bisection = fe621.util.computeAvgImpliedVolNewton(
49         data=spy_data1,
50         name='SPY',
51         rf=rf[data1_date][0],
52         current_date=data1_date,
53         tol=tol
54     )
55     end = datetime.now().timestamp()
56
57     # Computing time and number of options for Bisection

```



```

58     bisection_time = end - start
59     bisection_count = spy_vol_bisection.count(axis=0)[0]
60
61     # Building DataFrame, and saving to CSV
62     convergence_table = pd.DataFrame({
63         'Number of Input Options': [input_count, input_count],
64         'Number of Output Options': [newton_count, bisection_count],
65         'Number of Dropped Options': [input_count - newton_count,
66                                     input_count - bisection_count],
67         'Time Elapsed for Computation (s)': [newton_time, bisection_time],
68         'Average Time per Option (s)': [newton_time / input_count,
69                                       bisection_time / input_count]
70     })
71     convergence_table = convergence_table.T # Transposing so cols are methods
72     convergence_table.columns = ['Newton Method', 'Bisection Method']
73     convergence_table.to_csv('Homework 1/bin/imp_vol_convergence.csv')
74
75 if __name__ == '__main__':
76     compareConvergenceTime()

```

question.solutions/question_2.convergence.py

B.2.2 Implied Volatility Computation

```

1 from context import fe621
2
3 import numpy as np
4 import pandas as pd
5
6
7 # Defining dates
8 data1_date = '2019-02-06'
9 data2_date = '2019-02-07'
10
11 # Loading DATA1
12 spy_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/SPY',
13                                date=data1_date)
14 amzn_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/AMZN',
15                                date=data1_date)
16 vix_data1 = fe621.util.loadData(folder_path='Homework 1/data/DATA1/VIX',
17                                date=data1_date)
18
19 # Loading DATA2
20 spy_data2 = fe621.util.loadData(folder_path='Homework 1/data/DATA2/SPY',
21                                date=data2_date)
22 amzn_data2 = fe621.util.loadData(folder_path='Homework 1/data/DATA2/AMZN',
23                                date=data2_date)
24 vix_data2 = fe621.util.loadData(folder_path='Homework 1/data/DATA2/VIX',
25                                date=data2_date)
26
27 # Loading Risk-free rate (effective federal funds rate)
28 rf = pd.read_csv('Homework 1/data/ffr.csv')
29
30 # Tolerance level for optimization
31 tol = 1e-6
32
33 def computeImpVolatilities():
34     """Function to compute the implied volatilities for the SPY and AMZN option
35     chains, for all maturities. Computed implied volatilities are output to

```

```

36     CSV files.
37     """
38
39     # SP 500
40     spy_data1_vol = fe621.util.computeAvgImpliedVolBisection(
41                                     data=spy_data1,
42                                     name='SPY',
43                                     rf=rf[data1_date][0],
44                                     current_date=data1_date,
45                                     tol=tol)
46
47     # Saving to CSV
48     spy_data1_vol.to_csv('Homework 1/bin/spy_data1_vol.csv', index=False)
49
50     # AMZN
51     amzn_data1_vol = fe621.util.computeAvgImpliedVolBisection(
52                                     data=amzn_data1,
53                                     name='AMZN',
54                                     rf=rf[data1_date][0],
55                                     current_date=data1_date,
56                                     tol=tol)
57
58     # Saving to CSV
59     amzn_data1_vol.to_csv('Homework 1/bin/amzn_data1_vol.csv', index=False)
60
61 if __name__ == "__main__":
62     # Part 1 - Implied Volatility Computation
63     computeImpVolatilities()

```

question_solutions/question_2.imp_vol.py

B.2.3 Volatility Plots

```

1 from context import fe621
2
3 from mpl_toolkits.mplot3d import Axes3D
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import pandas as pd
7
8
9 # Loading implied volatility data from CSV files
10 spy_imp_vol = pd.read_csv('Homework 1/bin/spy_data1_vol.csv',
11                           index_col=False, header=0)
12 amzn_imp_vol = pd.read_csv('Homework 1/bin/amzn_data1_vol.csv',
13                             index_col=False, header=0)
14
15 # Defining date of DATA1
16 data1_date = '2019-02-06'
17
18
19 def plot2DVolsmile(data: pd.DataFrame, name: str, save_loc: str):
20     """Function to plot a 2D Volatility Smile for a given option chain.
21
22     Arguments:
23         data {pd.DataFrame} -- Input data containing implied volatilities.
24         name {str} -- Name of the underlying asset.
25         save_loc {str} -- Location (folder) to save the output image.
26     """
27

```

```

28 # Iterating through types of options for 2 separate put/call imp vol plots
29 for option_type_group in data.groupby('type'):
30     # Isolating current option type
31     option_type = option_type_group[0]
32
33     # Iterating through expiration dates for individual lines for each
34     for exp_date_group in option_type_group[1].groupby('expiration'):
35         # Isolating current expiration date
36         exp_date = exp_date_group[0]
37
38         # Sorting data to be ascending on 'strike'
39         plt_data = exp_date_group[1].sort_values(by='strike')
40
41         # Plotting strike vs implied vol
42         plt.plot(plt_data['strike'], plt_data['implied_vol'],
43                 label=('Maturity on ' + exp_date))
44
45     # Formatting plot
46     #=====
47
48     ax = plt.gca() # Get current axes
49
50     # Setting y ticks and label
51     ax.set_yticklabels(['{:.1%}'.format(i) for i in ax.get_yticks()])
52     ax.set_ylabel('Implied Volatility')
53     # Setting x ticks and label
54     ax.set_xticklabels(['$%i' % i for i in ax.get_xticks()])
55     ax.set_xlabel('Strike Price')
56
57     # Setting legend and setting plot dimensions to tight
58     plt.legend()
59     plt.tight_layout()
60
61     # Saving to file
62     full_option_type = 'Call' if (option_type == 'C') else 'Put'
63     fname = '_' .join([name, full_option_type, '2DVolSmile.png'])
64     plt.savefig(fname=(save_loc + '/' + fname))
65
66     # Closing plot for next one
67     plt.close()
68
69
70 def plot3DVolatilitySurface(data: pd.DataFrame, name: str, save_loc: str):
71     """Fuction to plot a 3D Volatility Surface for a given option chain.
72
73     Arguments:
74         data {pd.DataFrame} -- Input data containing implied volatilities
75         name {str} -- Name of the underlying asset.
76         save_loc {str} -- Location (folder) to save the output image.
77     """
78
79     # Iterating through types of options for 2 separate put/call imp vol plots
80     for option_type_group in data.groupby('type'):
81         # Isolating current option type
82         option_type = option_type_group[0]
83
84         # Isolating plot data
85         plot_data = option_type_group[1]
86
87         # Creating new column with time to maturity information for each option
88         ttm = plot_data.apply(lambda row: fe621.util.getTTM(

```

```

89         name=row.loc['name'],
90         current_date=data1_date),
91         axis=1)
92     # Converting TTM to days
93     ttm_days = ttm * 365
94
95     # Isolating data for each axis
96     x = np.array(ttm_days)
97     y = np.array(plot_data['strike'])
98     z = np.array(plot_data['implied_vol'])
99
100    # Plotting surface
101    fig = plt.figure()
102    ax = fig.gca(projection='3d')
103    ax.plot_trisurf(x, y, z, cmap='plasma')
104
105    # Formatting plot
106    #-----
107
108    # Setting x label
109    ax.set_xlabel('TTM (Days)')
110    # Setting y label
111    ax.set_ylabel('Strike Price ($)')
112    # Setting z label
113    ax.set_zlabel('Implied Volatility')
114
115    # Modifying z ticks to be percentages
116    ax.set_zticklabels(['{:,.0%}'.format(i) for i in ax.get_zticks()])
117
118    # Setting plot dimensions to tight
119    plt.tight_layout()
120
121    # Saving to file
122    full_option_type = 'Call' if (option_type == 'C') else 'Put'
123    fname = '_' .join([name, full_option_type, '3DVolSurface.png'])
124    plt.savefig(fname=(save_loc + '/' + fname))
125
126    # Closing plot for next one
127    plt.close()
128
129    if __name__ == '__main__':
130        # Plotting 2D Volatility Smile for AMZN and SPY option chains
131        plot2DVolSmile(data=amzn_imp_vol, name='AMZN',
132                        save_loc='Homework 1/bin/vol_smile/')
133        plot2DVolSmile(data=spy_imp_vol, name='SPY',
134                        save_loc='Homework 1/bin/vol_smile/')
135
136        # Plotting 3D Volatility Surface for AMZN and SPY option chains
137        plot3DVolatilitySurface(data=spy_imp_vol, name='SPY',
138                                save_loc='Homework 1/bin/vol_surface/')
139        plot3DVolatilitySurface(data=amzn_imp_vol, name='AMZN',
140                                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

```

1 from context import fe621
2
3 import numpy as np
4 import pandas as pd
5
6
7 def truncationErrorAnalysis():
8     """Function to analyze the truncation error of the Trapezoidal and Simpson's
9     quadrature rules.
10    """
11
12    # Objective function
13    def f(x: float) -> float:
14        return np.where(x == 0.0, 1.0, np.sin(x) / x)
15
16    # Setting values for N
17    N = np.power(10, np.arange(3, 8))
18
19    # Setting values for a
20    a = np.power(10, np.arange(2, 7))
21
22    trapezoidal_vals = np.ndarray((N.size, a.size))
23    simpsons_vals = np.ndarray((N.size, a.size))
24
25    # Building function approximation table, varying N and A
26    for i in range(0, N.size):
27        for j in range(0, a.size):
28            # Trapezoidal rule approximation
29            trapezoidal_vals[i, j] = fe621.numerical_integration \
30                .trapezoidalRule(f=f, N=N[i], start=-a[j], stop=a[j])
31            # Simpsons rule trunc approximation
32            simpsons_vals[i, j] = fe621.numerical_integration \
33                .simpsonsRule(f=f, N=N[i], start=-a[j], stop=a[j])
34
35    # Computing the absolute difference from Pi (i.e. trunc error)
36    # and casting to DataFrame
37    trapezoidal_df = pd.DataFrame(np.abs(trapezoidal_vals - np.pi))
38    simpsons_df = pd.DataFrame(np.abs(simpsons_vals - np.pi))
39
40    # Setting row and column names
41    trapezoidal_df.columns = ['N = ' + str(i) for i in N]
42    trapezoidal_df.index = ['a = ' + str(i) for i in a]
43    simpsons_df.columns = ['N = ' + str(i) for i in N]
44    simpsons_df.index = ['a = ' + str(i) for i in a]
45
46    # Saving to CSV
47    trapezoidal_df.to_csv(
48        'Homework 1/bin/numerical_integration/trapezoidal_trunc_error.csv',
49        header=True, index=True, float_format='%.8e'
50    )
51    simpsons_df.to_csv(
52        'Homework 1/bin/numerical_integration/simpsons_trunc_error.csv',
53        header=True, index=True, float_format='%.8e'
54    )
55
56
57 if __name__ == '__main__':

```

```

58 # Part 2 - Truncation Error Analysis
59 truncationErrorAnalysis()

```

question_solutions/question_3_trunc_error.py

B.3.2 Convergence Segment Analysis

```

1 from context import fe621
2
3 import numpy as np
4 import pandas as pd
5
6
7 def convergenceSegmentLimit():
8     """Function to compute the number of segments required for convergence of
9     various quadrature methods.
10    """
11
12    # Objective function
13    def f(x: float) -> float:
14        return np.where(x == 0.0, 1.0, np.sin(x) / x)
15
16    # Setting target tolerance level for termination
17    epsilon = 1e-3
18
19    # Using Trapezoidal rule
20    trapezoidal_result = fe621.numerical_integration.convergenceApproximation(
21        f=f,
22        rule=fe621.numerical_integration.trapezoidalRule,
23        epsilon=epsilon
24    )
25
26    # Using Simpson's rule
27    simpsons_result = fe621.numerical_integration.convergenceApproximation(
28        f=f,
29        rule=fe621.numerical_integration.simpsonsRule,
30        epsilon=epsilon
31    )
32
33    # Building DataFrame of results for output
34    results = pd.DataFrame(np.abs(np.array([trapezoidal_result,
35        simpsons_result])))
36
37    # Setting row and column names
38    results.columns = ['Estimated Area', 'Segments']
39    results.index = ['Trapezoidal Rule', 'Simpson\'s Rule']
40
41    # Saving to CSV
42    results.to_csv('Homework 1/bin/numerical_integration/convergence.csv',
43        header=True, index=True, float_format='%.8e')
44
45
46 if __name__ == '__main__':
47     # Part 3 - Convergence Analysis
48     convergenceSegmentLimit()

```

question_solutions/question_3_convergence.py

B.3.3 Arbitrary Function Convergence Segment Analysis

```

1 from context import fe621
2
3 import numpy as np
4 import pandas as pd
5
6
7 def arbitraryFunctionSegmentAnalysis():
8     """Function to analyze number of segments required for an arbitrary function
9     to converge under the Trapezoidal and Simpson's quadrature rules.
10    """
11
12    # Defining objective function
13    def g(x: float) -> float:
14        return 1 + np.exp(-1 * np.power(x, 2)) * np.cos(8 * np.power(x, 2/3))
15
16    # Setting target tolerance level for termination
17    epsilon = 1e-4
18
19    # Setting start and stop limits
20    start = 0
21    stop = 2
22
23    # Trapezoidal rule
24    trapezoidal_result = fe621.numerical_integration.convergenceApproximation(
25        f=g,
26        rule=fe621.numerical_integration.trapezoidalRule,
27        start=start,
28        stop=stop,
29        epsilon=epsilon
30    )
31
32    # Simpson's rule
33    simpsons_result = fe621.numerical_integration.convergenceApproximation(
34        f=g,
35        rule=fe621.numerical_integration.simpsonsRule,
36        start=start,
37        stop=stop,
38        epsilon=epsilon
39    )
40
41    # Building DataFrame of results for output
42    results = pd.DataFrame(np.abs(np.array([trapezoidal_result,
43        simpsons_result])))
44
45    # Setting row and column names
46    results.columns = ['Estimated Area', 'Segments']
47    results.index = ['Trapezoidal Rule', 'Simpson\'s Rule']
48
49    # Saving to CSV
50    results.to_csv('Homework 1/bin/numerical_integration/arb_convergence.csv',
51        header=True, index=True, float_format='%.8e')
52
53
54 if __name__ == '__main__':
55     # Part 4 - Arbitrary Function
56     arbitraryFunctionSegmentAnalysis()

```

question_solutions/question_3_arbitrary_area.py