Financial Engineering: Project 3

Jungwoo Yang

June 6, 2021

1 Introduction

European 콜옵션의 Black Scholes 방정식은 다음과 같이 정의된다.

$$C(S_t, t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)} \tag{1}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\frac{S_t}{K} + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right] \tag{2}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} \tag{3}$$

위 식에서 필요한 조건들은 현재 underlying 가격 S_t, strike price K, 현재 시점 t, 만기일 T, 무위험 이자율 r과 변동성 σ이다. 6가지의 인자들에서 바로 관찰이 되지 않는 변수는 변동성이기 때문에 변동성을 아는 것이 중요하다. 그동안은 변동성을 구하기 위해서 underlying의 historical price의 변동성을 구했었다. Historical price를 통해 변동성을 구하는 방법은 historical price를 구하기 위한 기간을 설정해야하고 기간 설정에 사람의 판단이 들어간다. 하지만 다행인 점은 옵션들은 시장에서 활발하게 거래가 되기 때문에 이론가의 출력에 해당하는 옵션 가격에 대한 정보가 있다.

Black Scholes의 방정식에서 변동성을 빼고 다른 인자들은 결정이 되어있기에 옵션 가격에 해당하는 변동성이 존재한다. 시장에서 옵션 가격을 관찰하고 관찰한 가격을 토대로 역으로 변동성을 계산하면 현시점에서의 underlying에 대한 시장의 기대 변동성을 구할 수 있다. 이렇게 구한 변동성을 내재 변동성(implied volatility)라고 한다. 또한 같은 underlying에 대해서도 다양한 strike price에 따른 옵션이 존재하기 때문에 한 시점에 만기일이 같은 옵션들일지라도 변동성이 같다는 보장이 없다. 따라서 만기일이 같은 옵션들의 가격에 따른 변동성을 관찰하는 것이 의미가 있을 것이다. 또한 같은 만기일이 아닌 옵션의 경우 더욱 다양한 변동성이 존재할 것이다. 내재 변동성을 관찰하면 더 이상 과거의 데이터에 의존하지 않고 underlying의 현 시점의 시장의 기대 변동성을 알 수 있다.

2 Black Scholes 방정식의 변동성 구하기

Black Scholes 방정식의 변동성을 구하는 방법에는 여러가지가 있다. 첫번째 방법은 Newton Raphson Method이다. Newton Raphson Method는 초기값 x_0을 정하고 현재점 $(x_0, f(x_0))$을 지나고 기울기가 $f'(x_0)$인 직선을 만든다. 직선이 x축과 교차하는 지점 x_1을 다시 다음 점으로 선택하는 방법이다. 즉 어떠한 함수를 해당점에서 일차식으로 근사를 하여 다음 점을 찾는 방식이다. 함수가 직선에 가까울 수록 아주 빠르게 해를 찾는다는 장점이 있다. 하지만 초기값이 잘못될 경우 원하지 않는 값을 출력하거나 수렴하지 않는다는 단점이 있다.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4}$$

Newton Raphson Method은 빠르다는 장점이 있지만 안정성 부분에서는 문제가 있을 수 있고 대상 함수의 일차 미분값을 알아야 한다는 단점이 있다. 그래서 Bisection Method를 사용하여 Black Scholes 방정식의 근을 찾도록 하였다. Bisection Method는 다음의 성질을 이용한다. $f(x)$가 x_l과 x_u에서 연속적이고 $f(x_l)$와 $f(x_u)$의 부호가 다르다면 x_l과 x_u 사이에서 $f(x_0) = 0$이 되는 점 x_0가 존재한다.

$$f(x_l)f(x_u) < 0 \tag{5}$$

그러한 점을 이용해 다음과 같은 코드를 통해 Black Scholes에서 변동성을 역으로 구하였다.

```
def cal_imvol_np(premium, S, K, T, r):
    if isinstance(premium, np.ndarray):
        bot, top = np.zeros(premium.shape[0]), \
                   3*np.ones(premium.shape[0])
    else:
        bot, top = 0, 3
    h = (bot+top)/2
    for _ in range(40):
        diff = BS_call_np(S, K, T, h, r) - premium
        bot = np.where(diff<0, h, bot)
        top = np.where(diff>=0, h, top)
        h = (bot+top)/2
    return h
```

Listing 1: cal_imvol_np()

변동성이 0 이하가 되는 경우는 없기에 initial lower bound를 0으로 설정을 하고 변동성이 3 이상이 되는 경우도 없었기에 initial upper bound를 3으로 설정했다. 초기값을 선택한 이후

에는 lower bound와 upper bound의 중간 h에서의 함수의 값을 확인한다. 만약 premium이 이론값보다 크다면 현재 선택한 변동성이 작다는 뜻이므로 h를 lower bound로 설정한다. 반대로 premium이 이론값보다 작다면 현재 선택한 변동성이 너무 크다는 뜻이므로 h를 upper bound로 설정한다. 빠른 계산을 하기 위해 threshold를 주기보다 횟수를 제한하고 numpy를 사용하여 병렬화를 하였다. 이러한 제한을 주지 않으면 많은 implied volatility를 계산할 때 속도가 너무 느려질 수 있기 때문이다. Newton Raphson Method는 많은 경우 좋은 성능을 보이지만 Bisection Method도 충분히 빠르다고 할 수 있다. 매 횟수 마다 찾는 범위를 절반으로 줄여 binary search와 동등한 $\mathcal{O}(\log n)$이 걸리게 된다. 예를 들어 x 좌표의 오차범위를 0.001로 하고자 하면 $\log_2 3,000 \approx 11.55$회 점화식을 돌리면 되고 오차범위를 0.000001로 하고자 하면 $\log_2 3,000,000 \approx 21.52$회 돌리면 된다. 총 40회 시행하도록 되어있다.

3 How to run

`python3 양정우hw103.py`

다음과 같이 순차적으로 모델이 나온다.

1. Penalized B-spline Curve로 보간한 Call Premium Curve
2. Penalized B-spline Curve로 보간한 Implied Volatility Curve
3. B-spline Surface로 보간한 Call Premium Surface
4. B-spline Surface로 보간한 Implied Volatility Surface

또한 위 모델에 해당하는 이미지가 현 디렉토리에 저장된다. (곡선의 경우 2021년 6월 11일로 되어있지만 실제로는 2021년 6월 7일 만기 콜 옵션이다)

- spxcall210611_curve.png
- spxcall210611_curve_imvol.png
- spxcall210611_210611_surface.png
- spxcall210611_210611_surface_imvol.png

4 Data Collection & Data Processing

4.1 옵션 데이터

옵션 데이터를 받을 때 중요한 조건들이 몇가지 있었다. 한 underlying에 대해 가능한 다양한 strike price로 거래가 되어야하고 가능한 많은 만기일이 있어야한다. 즉 거래가 많아야 하는 것이다. 그러한 옵션으로는 SPX만한 것이 없다고 생각이 된다. 따라서 S&P500 지수를 underlying으로 하는 옵션 SPX를 대상 데이터로 설정하였다. 옵션 데이터는 python의 yfinance library를 이용하였다. (`https://pypi.org/project/yfinance/`) 모든 데이터는 2021년 6월 5일에 받았으며(미국 기준 금요일 시장이 끝난후) 오늘이 6월 4일(금요일)이고 주식 시장이 끝난 후라고 가정하고 모든 값들을 계산하였다.

```
1 import yfinance as yf
2 sp500 = yf.Ticker("^SPX")
3 datelist = sp500.options
4 sp500.option_chain('2021-06-07')[0]
```

Listing 2: option data collecting

코드 2위와 같은 코드를 사용한 경우 `datalist`에는 만기일의 리스트가 리턴되고 4번째 줄에는 2021년 6월 7일을 만기로 하는 콜옵션들의 dataframe이 그림 1와 같이 리턴된다.

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	impliedVolatility	inTheMoney	contractSiz
0	SPXW210607C03375000	2021-06-04 14:30:39	3375.0	843.88	0.0	0.0	843.88	NaN	4.0	0	0.000010	True	REGULA
1	SPXW210607C03400000	2021-06-04 14:31:15	3400.0	818.52	0.0	0.0	818.52	NaN	2.0	0	0.000010	True	REGULA
2	SPXW210607C03475000	2021-06-04 14:32:08	3475.0	742.83	0.0	0.0	742.83	NaN	4.0	0	0.000010	True	REGULA
3	SPXW210607C03500000	2021-06-04 14:25:31	3500.0	718.75	0.0	0.0	718.75	NaN	1.0	0	0.000010	True	REGULA
4	SPXW210607C03660000	2021-06-04 13:56:52	3660.0	555.84	0.0	0.0	555.84	NaN	6.0	0	0.000010	True	REGULA
...	...	...	...	...	...	...	...	...	...	...	...	...	
122	SPXW210607C04575000	2021-06-04 13:34:36	4575.0	0.05	0.0	0.0	-0.25	-83.33333	1.0	13	0.250007	False	REGULA
123	SPXW210607C04580000	2021-06-04 13:31:50	4580.0	0.05	0.0	0.0	0.05	NaN	2.0	0	0.250007	False	REGULA
124	SPXW210607C04600000	2021-05-10 18:27:40	4600.0	0.36	0.0	0.0	0.00	0.00000	NaN	41	0.250007	False	REGULA
125	SPXW210607C04650000	2021-05-10 14:30:57	4650.0	0.15	0.0	0.0	0.00	0.00000	1.0	1	0.250007	False	REGULA
126	SPXW210607C04700000	2021-06-01 20:01:18	4700.0	0.05	0.0	0.0	0.00	0.00000	63.0	70	0.250007	False	REGULA

127 rows × 14 columns

Figure 1: 2021-06-07 calls

2021년 6월 5일에 받았을 때 존재하는 날짜들은 다음과 같다. (‘2021-06-07’, ‘2021-06-09’, ‘2021-06-11’, ‘2021-06-14’, ‘2021-06-16’, ‘2021-06-18’, ‘2021-06-21’, ‘2021-06-23’, ‘2021-06-25’, ‘2021-06-28’, ‘2021-06-30’, ‘2021-07-02’, ‘2021-07-06’, ‘2021-07-07’, ‘2021-07-09’,

'2021-07-16', '2021-07-23', '2021-07-30', '2021-08-20', '2021-08-31', '2021-09-17', '2021-09-30', '2021-10-15', '2021-10-29', '2021-11-19', '2021-11-30', '2021-12-17', '2021-12-31', '2022-01-21', '2022-02-18', '2022-03-18', '2022-03-31', '2022-04-14', '2022-05-20', '2022-06-17', '2022-12-16', '2023-12-15', '2026-03-20') 총 38개의 만기일이 있다. 각각의 만기일에 해당하는 옵션들을 저장하는 dataframe을 `./data/calls` 디렉토리에 저장하였다.

4.2 무위험 이자율

Black Scholes 방정식을 구할때 무위험 이자율이 필요하다. 이자율은 많이 쓰이는 3개월 만기 미국 국채를 사용했다(3-Month Treasury Bill: Secondary Market Rate (DTB3)). TBILL 데이터를 사용하기 위해서 pandas datareader library를 이용해 Federal Reserve Economic Data(FRED)에서 다운 받았다. 2021년 6월 6일에 데이터를 받았으며 제일 최근 존재하는 값을 사용했다(0.01%). 다음과 같은 코드를 통해 받고 `./data/TBILL_d210606.h5`에 저장되어 있다.

```
import pandas_datareader as pdr
t_df = pdr.DataReader('DTB3', 'fred', start='2021-05-01')
```

Listing 3: TBILL data

4.3 실험 환경

Underlying의 가격을 결정해야하는데 2021년 6월 4일 마감 가격인 $4229.89을 underlying 가격으로 결정했다.

```
today = date(2021, 6, 4)
S = 4229.89 #S&P500 2021-06-04 close
```

Listing 4: setting

SPX가 거래량이 많긴 하지만 underlying 만큼 매 시점마다 모든 strike price에 대한 가격이 존재하는 것이 아니다. 따라서 2021년 6월 4일에 거래된 콜가격으로만 제한했다. 이러한 가정으로 2021년 6월 7일이 만기인 콜옵션을 scatter plot으로 보이면 다음과 같다. 그림 2는 마지막 가격이 존재하는 모든 점들이고 그림 3은 6월 4일에 거래된 모든 점들이다.

4.4 내재 변동성 변환

그림 3의 콜 가격을 내재 변동성으로 변환해 그림 4와 같다.

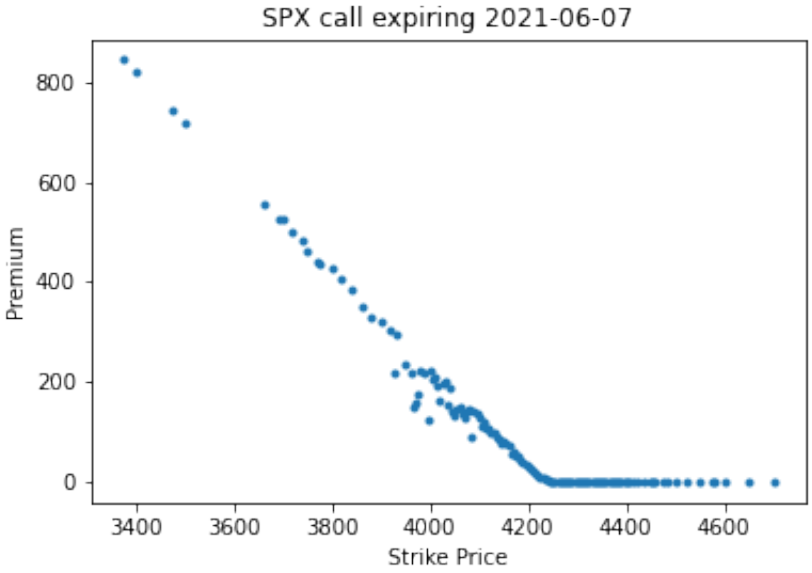

Figure 2: SPX call 2021-06-07, 모든 거래

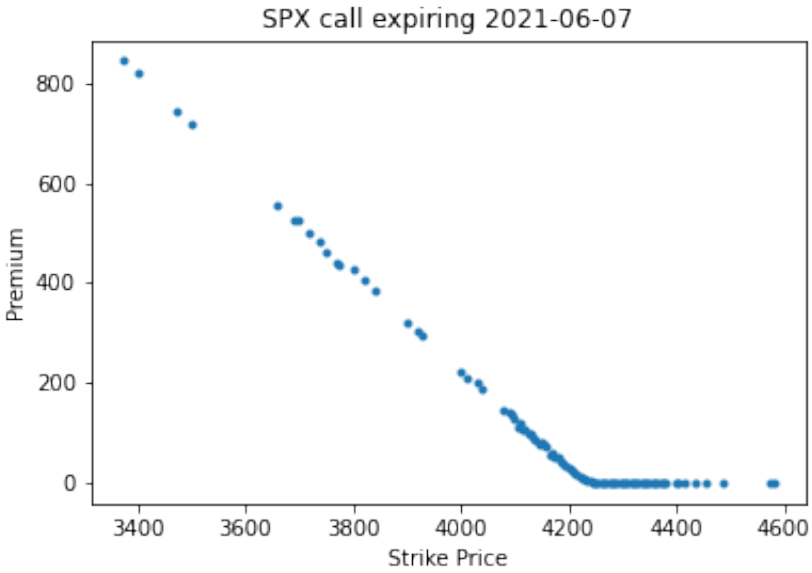

Figure 3: SPX call 2021-06-07, 6월 4일에 거래된 콜가격

5 곡선 보간법

그림 3과 그림 4와 같이 가격이 존재하는 콜 옵션들의 점들을 찍을 수 있지만 점이 없거나 점들의 해당 점이 다른 점들의 추이와 비교해 outlier인지 아닌지가 알기 힘들다. 따라서 전 strike price에서 콜옵션의 가격을 알고 싶으면 존재하는 점들을 보간하는 방법이 있을 것이다.

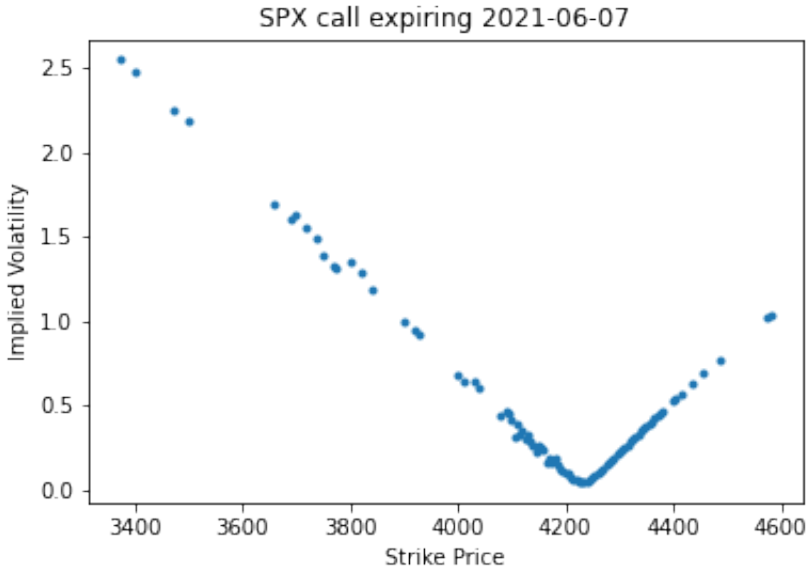

Figure 4: SPX call 2021-06-07, 6월 4일에 거래된 콜가격, 내재 변동성으로 변환

5.1 Bézier Curve

첫번째 사용한 방법은 Bézier Curve이다. 점 $P_0, P_1, ..., P_n$가 정의될때 Bézier Curve는 다음과 같이 재귀적으로 정의된다.

$$B_{P_0}(t) = P_0 \tag{6}$$

$$\begin{aligned} B(t) &= B_{P_0,P_1,...,P_n}(t) \\ &= (1-t)B_{P_0,P_1,...,P_{n-1}}(t) + tB_{P_1,P_2,...,P_n}(t) \end{aligned} \tag{7}$$

이렇게 보간한 곡선은 $n-1$차 곡선이 된다.

Bézier Curve를 구하는 방법은 여러가지가 있지만 De Casteljau의 방법을 사용하여 다음과 같은 코드를 통해 계산하였다.

```
def deCasteljau(i, j, u, D):
    if D[i][j]:
        return D[i][j]
    p0 = deCasteljau(i-1, j, u, D)
    p1 = deCasteljau(i-1, j+1, u, D)
    D[i][j] = ((1-u)*p0[0]+u*p1[0], (1-u)*p0[1]+u*p1[1])
    return D[i][j]

def bezier_curve(P, num_points= 100):
    points = []
    n = len(P)-1
```

```
    for u in np.linspace(0, 1, num_points):
        D = [[None]*(n+1) for _ in range(n+1)]
        for j in range(n+1):
            D[0][j] = P[j]
        points.append(deCasteljau(n, 0, u, D))
    return points
```

Listing 5: De Casteljau

De Casteljau의 방법은 Bézier Curve의 다음과 같은 속성을 통해 구하는 방법이다. 점 $P_{0,j}$가 j번째 초기점이라면,

$$P_{i,j} = (1-u)P_{i-1,j} + (u)P_{i-1,j+1} \quad \begin{cases} i = 1, 2, ..., n \\ j = 0, 1, ..., n-i \end{cases} \tag{8}$$

하지만 해당하는 점을 naive하게 계산하면 exponential time이 걸리기 때문에 dynamic programming을 사용해 곡선을 구하였다. 양정우`hw103.py`에 Bézier Curve를 그리는 코드는 있지만 시간이 촉박해 그림으로 output하는 코드는 추가하지 못했다. 2021년 6월 7일 만기인 SPX를 보간하면 그림 5가 그려진다.

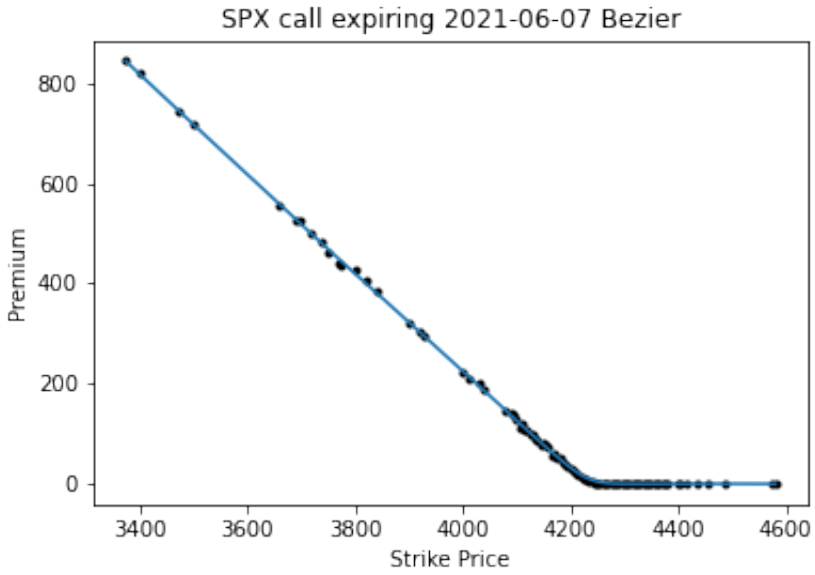

Figure 5: SPX call 2021-06-07, 6월 4일에 거래된 콜가격, Bézier Curve로 보간

다음과 같은 Bézier Curve를 그리는 코드를 추가하기면 하면 Bézier Curve가 그려진다.

```
df = pd.read_hdf('./data/calls/call_s210607_d210606.h5')
imvol_list = get_imvol_list(date(2021, 6, 7), df, moneyness=False, imvol
    =False)
```

```
X, Y = list(zip(*imvol_list))
xx, yy = list(zip(*bezier_curve(imvol_list)))
plt.scatter(X, Y, marker='.', c='k')
plt.plot(xx, yy)
plt.xlabel("Strike Price")
plt.ylabel("Premium")
plt.title("SPX call expiring 2021-06-07 Bezier")
plt.savefig("./SPX_call_2021_06_07_bezier.png")
```

Listing 6: Bézier 그림 output 방법

내재 변동성을 구한 후 위 코드로 Bézier Curve를 그리면 그림 6과 같이 된다.

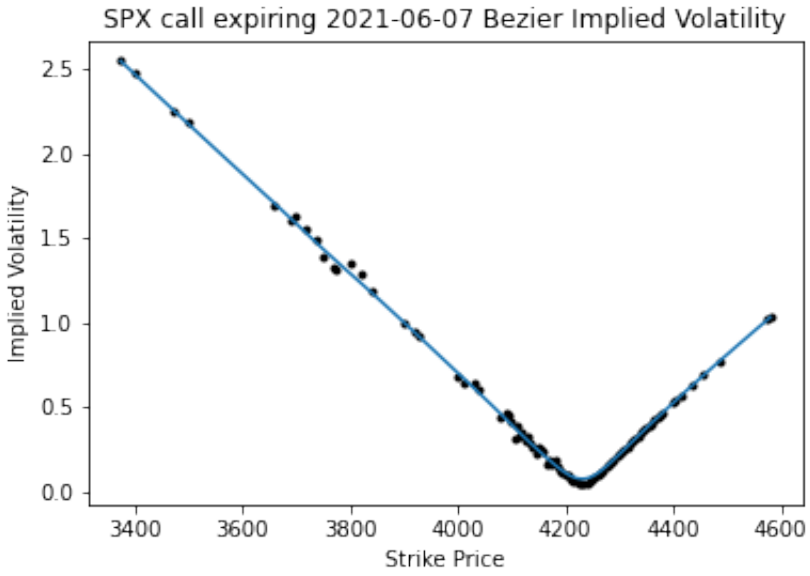

Figure 6: SPX call 2021-06-07, 6월 4일에 거래된 imvol, Bézier Curve로 보간

5.2 B-Spline

Bézier Curve를 그렸지만 무재정 조건을 추가하기 위해 B-spline 보간법을 추가했다. B-spline을 그리고 코드를 짜는데 금융공학VII [2]을 참고했다. B-spline 곡선은 order n 곡선을 그릴 경우, 다음과 같은 성질을 만족하는 곡선이다. 먼저 knots $t_0, t_1, t_2, ..., t_n$을 정한

뒤에 다음과 같은 식을 만족시키는 곡선이 spline이다.

$$B_{i,n}(x) = \begin{cases} 0 & \text{if } x < t_i \text{ or } x \geq t_{i+n} \\ \text{nonzero} & \text{otherwise} \end{cases} \tag{9}$$

$$\sum_i B_{i,n}(x) = 1 \tag{10}$$

그렇게 하면 order 1의 경우는

$$B_{i,1}(x) = \begin{cases} 1 & \text{if } t_i \leq x < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$\tag{12}$$

이된다. 또한 더 큰 order의 splines들은 다음과 같이 재귀적으로 정의된다.

$$B_{i,k+1}(x) := w_{i,k}(x)B_{i,k}(x) + [1 - w_{i+1,k}(x)]B_{i+1,k}(x), \tag{13}$$

$$\text{where } w_{i,k} := \begin{cases} \dfrac{x - t_i}{t_{i+k} - t_i}, & t_{i+k} \neq t_i \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

마지막으로 구한 B-splines들의 선형 결합을 통해 spline function을 나타낼 수 있다.

$$S_{n,t}(x) = \sum_i \alpha_i B_{i,n}(x) \tag{15}$$

Cublic B-spline을 구하는 코드는 다음과 같다.

```
def b_spline_base(P, x):
    """P: control points"""
    delta = np.mean(np.diff(P))
    n_P = len(P)
    t = [P[0]-delta*3, P[0]-delta*2, P[0]-delta*1]+\
            np.linspace(P[0],P[-1], n_P-2).tolist()+\
            [P[-1]+delta*1, P[-1]+delta*2, P[-1]+delta*3]
    B = [[0.0]*(4) for _ in range(n_P+4)]

    for i in range(len(t)):
        if t[i]<=x and x<t[i+1]:
            B[i][0]=1
            break
    for k in range(1, 4):
```

```
        for i in range(len(t)-k-1):
            B[i][k]= ((x-t[i])/(t[i+k]-t[i]))*B[i][k-1] + ((t[i+k+1]-x)
   /(t[i+k+1]-t[i+1]))*B[i+1][k-1]
    return [B[i][3] for i in range(len(B))]
```

Listing 7: Cubic B-Spline

입력되는 점들을 바탕으로 knots를 정의하였으며 마지막은 order 4의 x점에서의 b-spline 값을 리턴한다. Cubic B-spline이 정의되면 α 값을 구하여야한다. α값을 구하기 위해서는 다음과 같은 목적함수를 최소화해 구할 수 있다.

$$\arg\min_{\alpha} \quad \sum_{i}(y_i - S(x_i))^2 \tag{16}$$

즉 위 식은 B-spline들을 선형결합으로 만든 점 $\hat{y}_i$과 y_i의 거리의 제곱의 합을 최소화하는 것이다. 또한 곡률을 추가적으로 목적함수에 추가하여 평활하게 만들수 있다 [1].

$$\arg\min_{\alpha} \quad \sum_{x}(y_i - S(x_i))^2 + \lambda \sum_{i}(S''(x_i)) \tag{17}$$

또한 다음과 같은 무재정 조건을 추가하였다.

1. 콜옵션가격 $C(K)$는 행사가격 K에 대해 볼록하다.
2. 콜옵션가격 $C(K)$는 우하향한다.
3. 모든 구간에서 일차 도함수 $C'(K)$는 $-e^{-r\tau}$보다 크다.
4. τ가 0이 아니면 $C(K) > 0$이다.

5.2.1 Penalized B-spline Curve Code

그리하여 위를 종합해 작성한 코드는 다음과 같다.

```
def penalized_B_spline(P, tau,lam=0.1):
    _x, _y = list(zip(*P))
    _y = np.array(_y)
    bases = [b_spline_base(_x, i) for i in _x]
    bases = np.array(bases)

    m = gp.Model()
    n_P = len(P)
    x_c = np.linspace(_x[0], _x[-1], 1000)
    c_bases = [b_spline_base(_x, i) for i in x_c]
```

```
    alpha = m.addMVar(len(bases[0]), lb=-GRB.INFINITY, ub = GRB.INFINITY
    , name='alpha')
    BB = bases.transpose()@bases
    v = -2*bases.transpose() @_y
    D = np.diff(np.eye(len(_y)), 2).transpose() #second derivative
    dev = lam*bases.transpose()@D.transpose()@D@bases
    obj = _y@_y + alpha@v+ alpha@BB@alpha + alpha@dev@alpha
    m.setObjective(obj)

    c_bases = np.array(c_bases)
    D1 = np.diff(np.eye(len(x_c)), 1).transpose() #first derivative
    D2 = np.diff(np.eye(len(x_c)), 2).transpose() #second derivative
    #slope downward constraint
    D1_C = D1@c_bases
    m.addConstrs(D1_C[i]@alpha<=0 for i in range(D1.shape[0]))
    #slope shoud be bigger than e^(-rt)
    D1_C_d = -D1@c_bases/(x_c[1]-x_c[0])
    m.addConstrs(D1_C_d[i]@alpha-math.exp(-rf_rate*tau)<=0 for i in
    range(D1.shape[0]))
    #convex constraint
    D2_C = -D2@c_bases
    m.addConstrs(D2_C[i]@alpha<=0 for i in range(D2.shape[0]))
    #bigger than 0
    m.addConstr(c_bases[-1]@alpha>=0)
    m.optimize()
    return m
```

Listing 8: Penalized B-Spline

Gurobi Optimizer를 사용하였다. 또한 위와 같이 조건을 추가하기 위해 원래의 정의역에 많은 점 `x_c`을 만들어 조건을 추가할 수 있다. 조건 3번은 먼저 $C(K)$가 볼록하고 첫번째 두 점에서의 기울기가 $-e^{-r\tau}$이상이면 만족하게 된다. Discrte한 점에서 조건을 추가하였기 때문에 `D1`과 `D2`를 통해 1차 도함수, 2차 도함수를 구할 수 있다. 2차 도함수의 경우, 아래와 같은 식을 통해 근접하게 구할 수 있다.

$$\frac{(y_{i+2}-y_{i+1})-(y_{i+1}-y_i)}{(x_{i+2}-x_{i+1})-(x_{i+1}-x_i)} = \frac{y_{i+2}-2y_{i+1}+y_i}{x_{i+2}-2x_{i+1}+x_i}$$

5.2.2 결과

그림 7은 만기가 2021년 6월 7일인 SPX를 Penalized B-Spline을 통해 보간했을 때이다. (spxcall210611_curve.png) 원래 코드는 6월 11일 만기로 적혀있는데 실제로는 6월 7일 만기 콜옵션들이다.

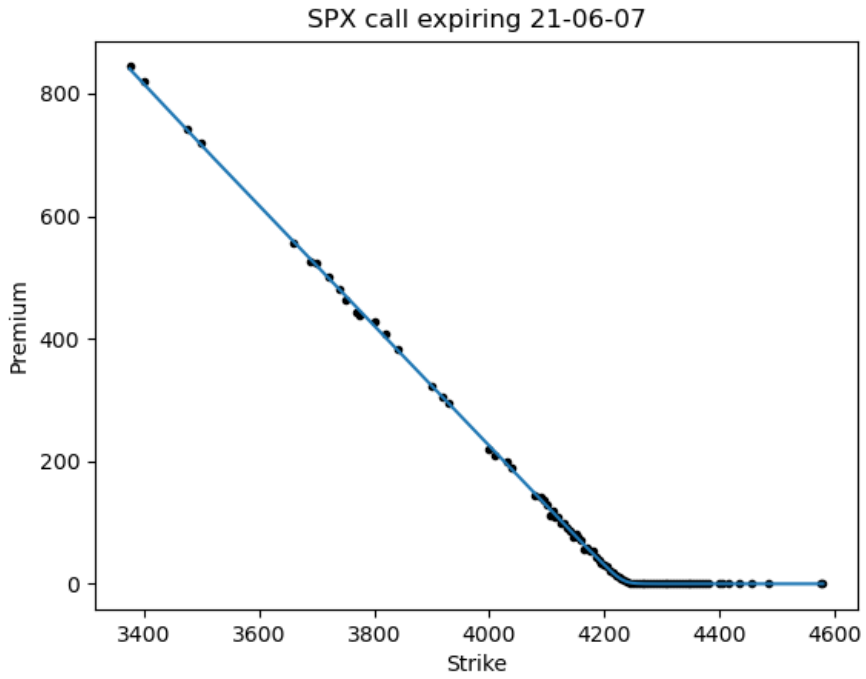

Figure 7: SPX call 2021-06-07, 6월 4일에 거래된 Premium, B-Spline Curve로 보간

그림 8은 만기가 2021년 6월 7일인 SPX를 Penalized B-Spline을 통해 보간한 후 내재변동성을 구했을 때의 곡선이다. (spxcall210611_curve_imvol.png) 원래 코드는 6월 11일 만기로 되어있는데 실제로는 6월 7일 만기 콜옵션이다.

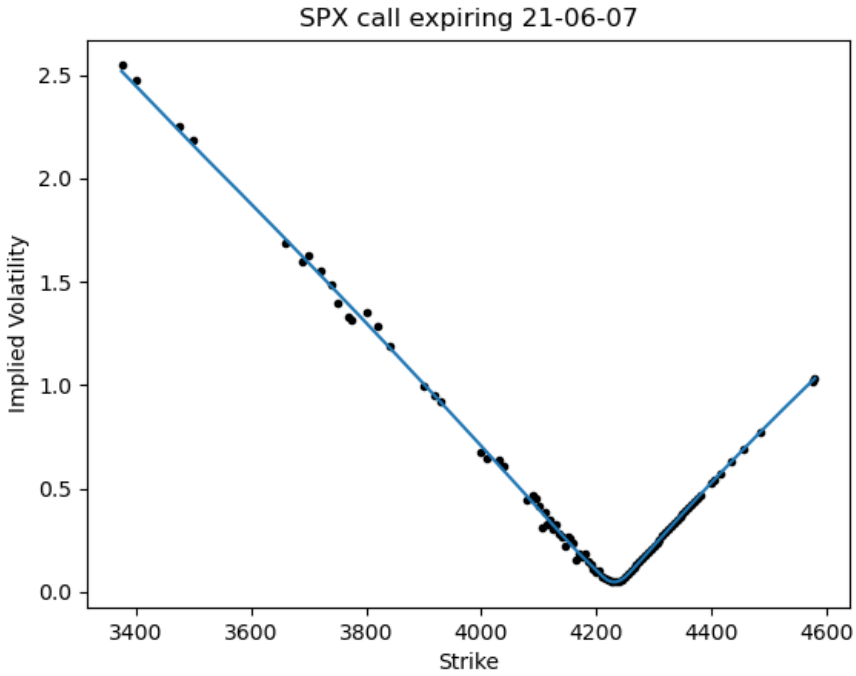

Figure 8: SPX call 2021-06-07, 6월 4일에 거래된 imvol, B-Spline Curve로 보간

6 Implied Volatility Surface

지금까지는 만기일을 고정시키고 strike price에 대해서 보간을 한 것이다. 하지만 만기일에 따라서 각각의 곡선이 있고 strike price S를 x축으로 만기 τ를 y축으로하고 premium이나 implied volatility를 z축으로 하는 곡면을 구할 수 있을 것이다.

6.1 B-Spline Surface

B-Spline Surface는 다음과 같이 정의 된다.

$$S(x,y) = \sum_j \sum_i \alpha_{i,j} B_i(x) B_j(y) \tag{18}$$

$\alpha_{i,j}$ 값을 구하기 위해서는 다음과 같은 목적함수를 최소화하여 구할 수 있다.

$$\arg\min_{\alpha} \quad \sum_j \sum_i (z_{i,j} - S(x_i, y_j))^2 \tag{19}$$

곡선에 대해서는 곡률에 대한 제한을 두었지만 곡면에서는 주지 않았다. 다음과 같은 무재정 조건을 추가하였다.

1. 잔여기간 τ가 고정되었을 때 콜옵션가격 $C(K,\tau)$는 행사가격 K에 대해 볼록하다.

2. 잔여기간 τ가 고정되었을 때 콜옵션가격 $C(K,\tau)$는 우하향한다.

3. 잔여기간 τ가 고정되었을 때 모든 구간에서 일차 도함수 $C'(K,\tau)$는 $-e^{-r\tau}$보다 크다.

4. τ가 0이 아니면 $C(K,\tau) > 0$이다.

5. 각 행사가격 K에 대해서, 잔여기간 τ가 증가함에 따라 $C(K,\tau)$ 는 감소하지 않는다.

6.2 B-Spline Surface Code

```
def gen_b_spline_surface():
    points3D= get3Dpoints()
    _x, _y, _z = [], [], []
    for p in points3D:
        t_x, t_y, t_z = list(map(list, zip(*p)))
        _x.append(t_x)
        _y.append(t_y)
        _z.append(t_z)
    tmp_x = []
    for i in _x:
        tmp_x.extend(i)
    tmp_y = []
    for i in _y:
        tmp_y.extend(i)

    mesh_tau = sorted(list(set(tmp_y)))[:4]
    mesh_strike = sorted(list(set(tmp_x)))[80:-77]

    bases_st = [b_spline_base(mesh_strike, i) for i in mesh_strike]
    bases_tau = [b_spline_base(mesh_tau, i) for i in mesh_tau]

    bases_3D = []
    _xx,_yy,_zz = [], [], []
    for i, _t in enumerate(mesh_tau):
        for j, _s in enumerate(mesh_strike):
            if _s in _x[i]:
                bases_3D.append(np.kron(bases_st[j], bases_tau[i]))
                idx = _x[i].index(_s)
                _zz.append(_z[i][idx])
                _xx.append(_x[i][idx])
```

```
                _yy.append(_y[i][idx])
    bases_3D = np.array(bases_3D)
    _xx, _yy, _zz = np.array(_xx), np.array(_yy), np.array(_zz)

    c_x = np.linspace(mesh_strike[0], mesh_strike[-1], 100)
    c_y = np.linspace(mesh_tau[0], mesh_tau[-1], 10)

    c_bases_st = [b_spline_base(mesh_strike, i) for i in c_x]
    c_bases_tau = [b_spline_base(mesh_tau, i) for i in c_y]

    c_bases_3D = []
    for i in range(len(c_y)):
        for j in range(len(c_x)):
            c_bases_3D.append(np.kron(c_bases_st[j], c_bases_tau[i]))
    c_bases_3D = np.array(c_bases_3D)

    m = gp.Model()
    alpha = m.addMVar(bases_3D[0].shape[0], lb=-GRB.INFINITY, ub=GRB.
INFINITY, name='alpha')

    BB = bases_3D.transpose()@bases_3D
    v = -2*bases_3D.transpose()@_zz
    obj = _zz@_zz + alpha@v + alpha@BB@alpha
    m.setObjective(obj)
    c_bases_3D = np.array(c_bases_3D)

    D1 = np.diff(np.eye(len(c_x)), 1).transpose()
    D2 = np.diff(np.eye(len(c_x)), 2).transpose()

    for i in range(len(c_y)):
        cur_bases = c_bases_3D[(i)*len(c_x):(i+1)*len(c_x)]

        D1_C = D1@cur_bases
        m.addConstrs(D1_C[j]@alpha<=0 for j in range(D1.shape[0]))

        D1_C_d = -D1@cur_bases/(c_x[1]-c_x[0])
        m.addConstrs(D1_C_d[j]@alpha-math.exp(-rf_rate*c_y[i])<=0 for j
in range(D1.shape[0]))

```

```
68          D2_c = -D2@cur_bases
69          m.addConstrs(D2_c[j]@alpha<=0 for j in range(D2.shape[0]))
70
71          m.addConstr(cur_bases[-1]@alpha>=0)
72
73      D1_y = np.diff(np.eye(len(c_y)), 1).transpose()
74      for i in range(len(c_x)):
75          cur_bases = c_bases_3D[i::len(c_x)]
76          D1_CY = D1_y@cur_bases
77          m.addConstrs(D1_CY@alpha>=0 for j in range(D1_y.shape[0]))
78
79      m.optimize()
```

Listing 9: Penalized B-Spline

곡선과 많은 부분 비슷한 코드를 가지고 있지만 line 73-line77을 통해 조건 5번을 gurobi에게 넘길 수 있다.

6.2.1 결과

그림 9은 만기가 2021-06-07, 2021-06-09, 2021-06-11, 2021-06-14인 SPX Premium를 B-Spline을 통해 보간했을 때의 곡면이다

그림 10은 만기가 2021-06-07, 2021-06-09, 2021-06-11, 2021-06-14인 SPX Premium B-Spline을 통해 보간하고 implied volatility를 구했을 때의 곡면이다.

7 Conclusion

B-spline curve는 잘 구하였ekrh 생각이 되지만 나머지는 시간 부족으로 아쉬움이 많이 남는 프로젝트였다. B-Spline Surface가 위의 점들에 한해서는 잘 구해졌지만 다른 점들에 대해서는 gurobi에서 $x^\intercal Qx + ax$에서 Q가 positive semi-definite이 아니어서 최적화 문제가 non-convex라고 불평을 했다. B-spline surface에서 Q의 경우 훨씬 더 큰 matrix이기도 하고 더 많은 0을 가지고 있기 때문이지 않을까 싶다. 또한 가격을 2021년 6월 4일 종가로 고정을 했는데 하루안에 S&P500 index의 변화에 따라 ill-condition으로 바뀐 콜옵션들도 있을 것이라 생각이 된다. 각각의 콜 옵션이 거래된 시간에 따라 underlying의 가격을 바꾸어 implied volatility를 구한뒤 다시 종가에 따라 premium을 구하였으면 되지 않을까 싶다. 또한 ill-condition을 바꾸는 다른 방법으로는 각각의 만기일마다 B-spline curve를 그리고 점들을 추출한 후 그 점들에 대해 B-spline surface을 구하면 더욱 안정적으로 B-spline surface를

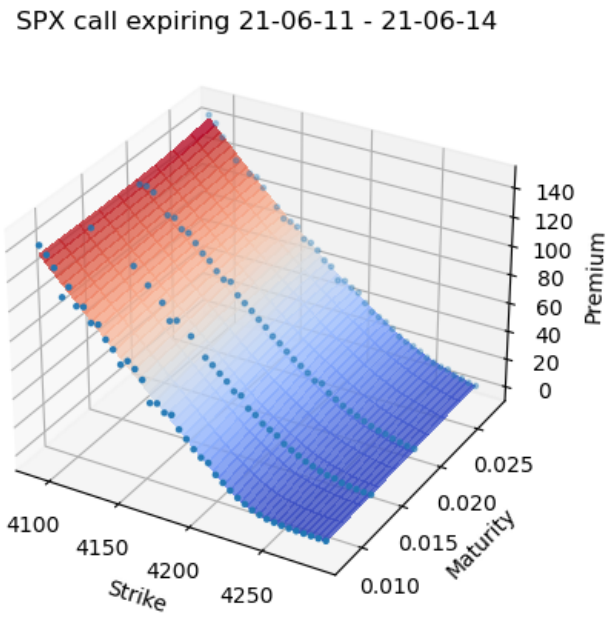

Figure 9: SPX call 2021-06-07 - 2021-06-14, 6월 4일에 거래된 Premium, B-Spline Surface로 보간

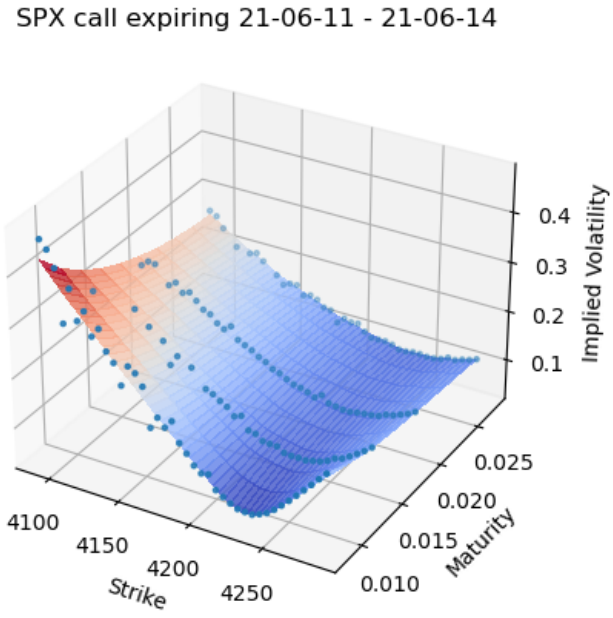

Figure 10: SPX call 2021-06-07 - 2021-06-14, 6월 4일에 거래된 imvol, B-Spline Surface로 보간

그릴 수 있을 것으로 추정한다. 또한 Bézier curve를 그릴 때 들어온 점들을 control point로 가정하고 그렸었는데 fitting을 해서 control point를 구하여 Bézier curve와 Bézier surface를 구하는 것도 해볼 수 있을 것이다.

References

[1] Paul HC Eilers, Brian D Marx, et al. Flexible smoothing with b-splines and penalties. *Statistical science*, 11(2):89–121, 1996.

[2] 최병선. 금융공학 vii: Scientific computing for finance and economics, 2018.