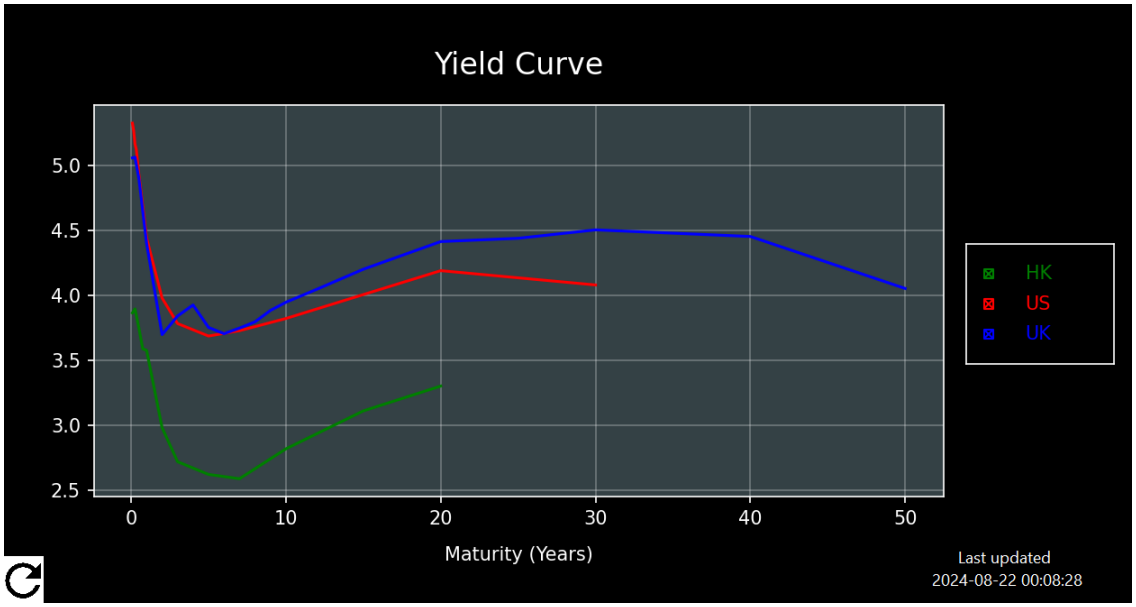
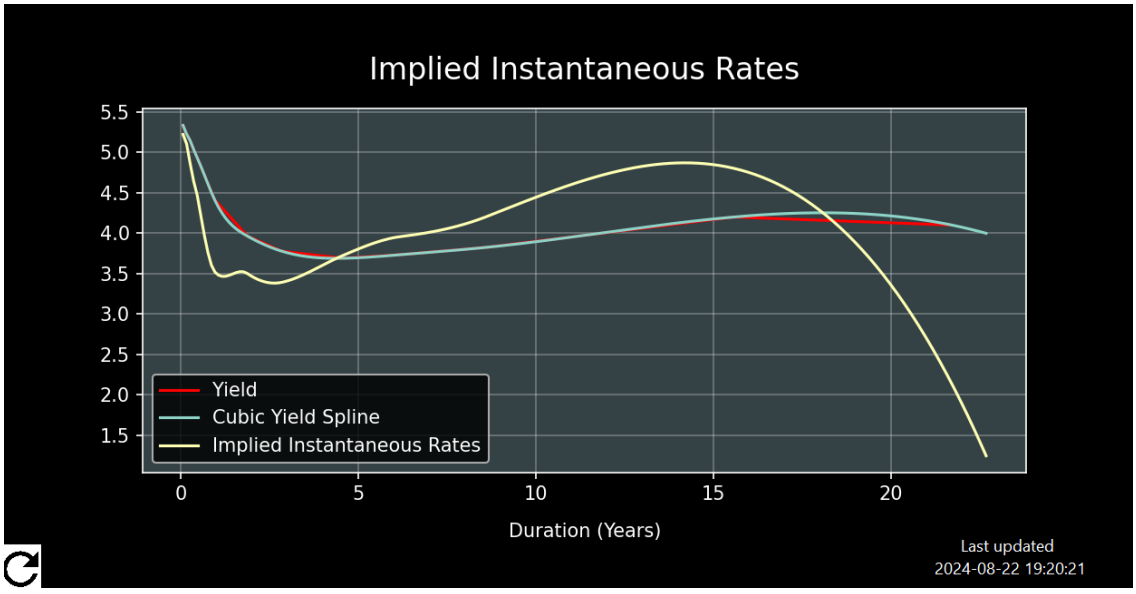
Volatility Surface

In this project, I attempt to build an arbitrage-free Black-Scholes volatility surface from liquid SPX options, web-scraped from yahoo finance.

One of the inputs we need in the Black-Scholes model is the annualised risk-free rate, which we attempt to estimate using treasury bond data found on either World Government Bonds or CNBC. Treasury bonds with maturity > 1 year have semi-annual coupons. Bonds with the same maturity but different coupon rates have different yield to maturity because of the difference in duration, therefore we use durations instead of maturities to remove this difference. The steps are as follows:

1. Find yield to maturity for coupon bonds with maturity T and duration D
2. Use these yields as proxies for the yields of zero-coupon bond with maturity D
3. Fit a cubic spline through the given points
4. Use the value of the cubic spline at the time to maturity of our option as the constant interest rate r in the Black-Scholes model





After scraping SPX option data using yfinance and BeautifulSoup, we filter the options using several criteria, including moneyness, time since last traded, non-zero bid and ask prices etc. We only use out-of-the-money options, as they are more actively traded due to their low premium and the demand for out-of-the-money puts for hedging. We then convert the put prices into call prices using the Put-Call Parity, and only work with call prices from now on.

$$C_t - P_t = S_t - Ke^{-r(T-t)}$$

The Put-Call Parity is not model-dependent, and hence not Black-Scholes specific. The payoffs of a European call and put are $(S_T - K)^+$ and $(K - S_T)^+$ respectively. Therefore,

$$\begin{aligned} C_T - P_T &= (S_T - K)^+ - (K - S_T)^+ \\ &= S_T - K \end{aligned}$$

and we can find a replicating portfolio for a portfolio consisting of a long call and short put, namely, borrowing Ke^{-rt} and buying the underlying stock at time 0. At time t, the position is worth $S_t - Ke^{-r(T-t)}$. Therefore, by no-arbitrage, the values of the two portfolios must be identical at every $t \in [0, T]$.

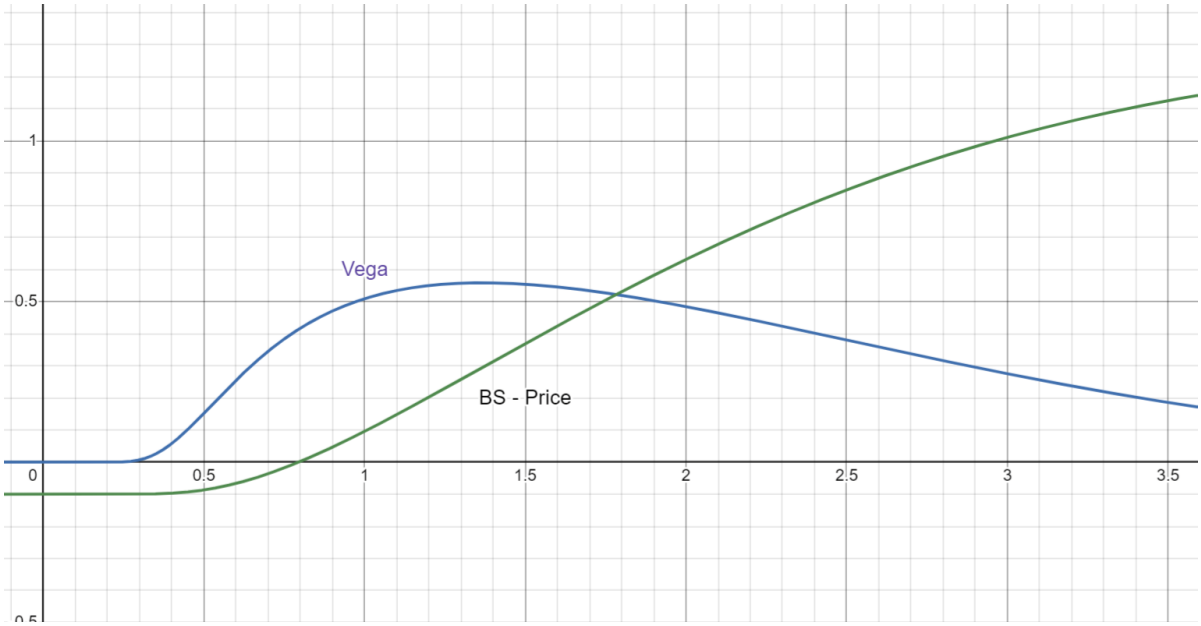
For each maturity, we use the Newton-Raphson numerical method to solve for the root σ that gives

$$BS(\sigma) - Market\ Price = 0$$

with initial guess

$$\sigma_{initial} = \sqrt{\left|\frac{2}{T}\log\left(\frac{S}{K}\right) + 2r\right|}$$

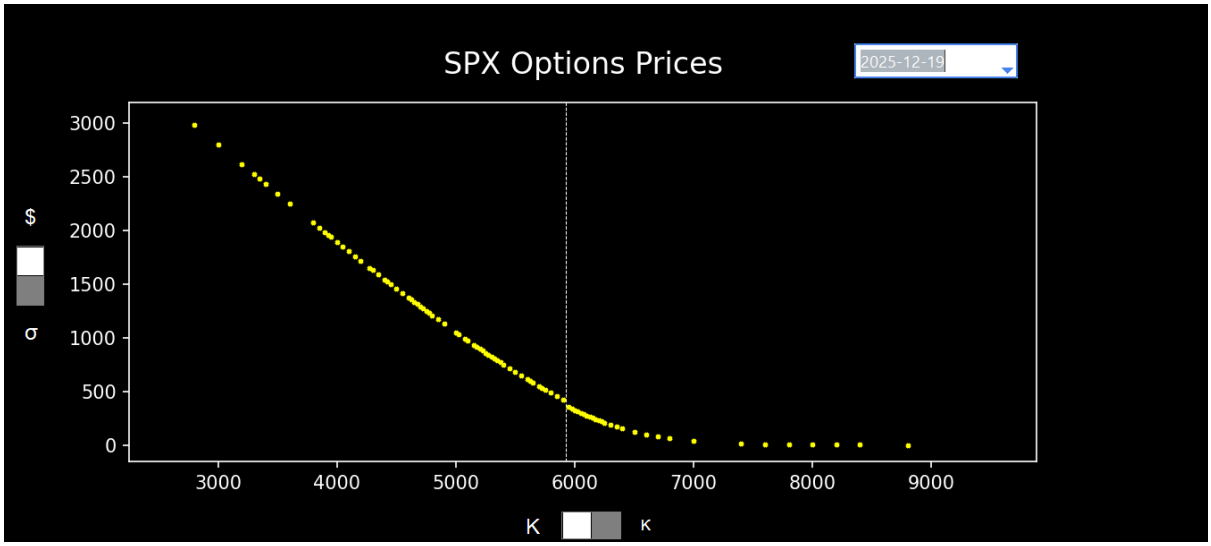
which is the inflection point on the graph of price against volatility and the maximum point on the graph of vega against volatility, to ensure that the answer converges. The target error in the algorithm is set to 10^{-5}, such that the process terminates when the estimated price and market price differ by no more than 10^{-5}.

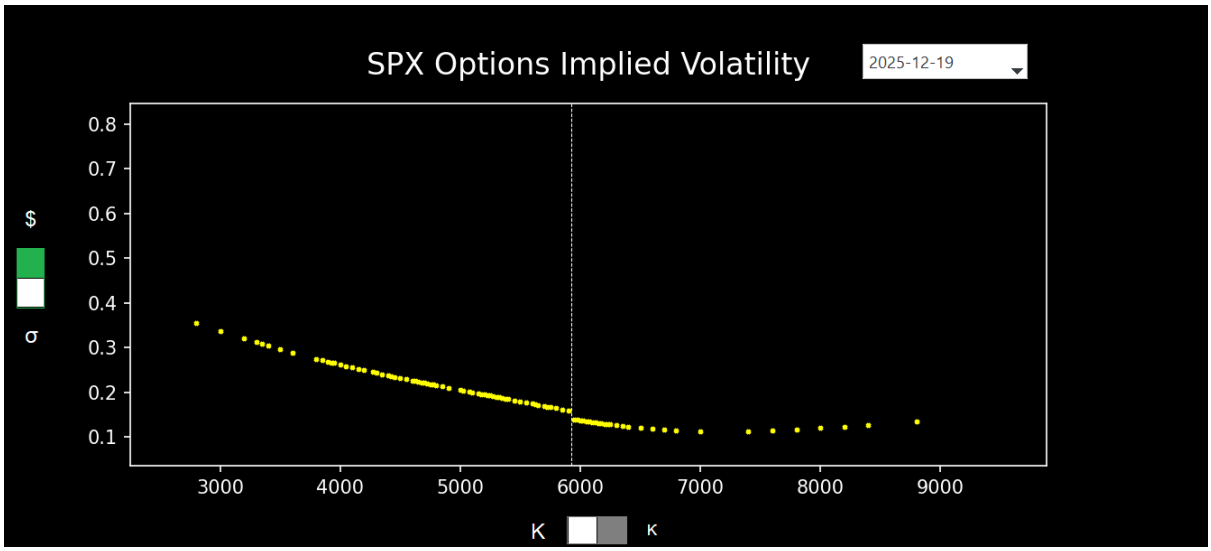



```python
def option_price_vega(option_type: str, S, K, T, r, sigma):
    d1 = math.log(S/K) / (sigma*math.sqrt(T)) + (r/sigma + sigma/2) * math.sqrt(T)
    d2 = d1 - sigma * math.sqrt(T)
    vega = S * math.sqrt(T) * norm.pdf(d1)
    if option_type == 'Call':
        price = S * norm.cdf(d1) - K * math.exp(-r * T) * norm.cdf(d2)
    elif option_type == 'Put':
        price = K * math.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
    else:
        return None
    return [price, vega]

def newton_raphson(row, price_type='mid'):
    target_error = 10 ** (-5)
    r = yield_spline(row['T']) / 100
    guess = math.sqrt(abs(2*r + 2*math.log(self.spot/row['strike'])/row['T']))
    if row['type'] in ['Call', 'Put']:
        if price_type == 'mid':
            target = (row['bid'] + row['ask'])/2
        else:
            target = row[price_type]
        try:
            while True:
                [price, vega] = option_price_vega(row['type'], self.spot,
                                                  row['strike'], row['T'], r, guess)
                if abs(price - target) <= target_error:
                    return guess
                else:
                    guess -= (price - target) / vega
        except ZeroDivisionError:
            return None
    else:
        return None
```

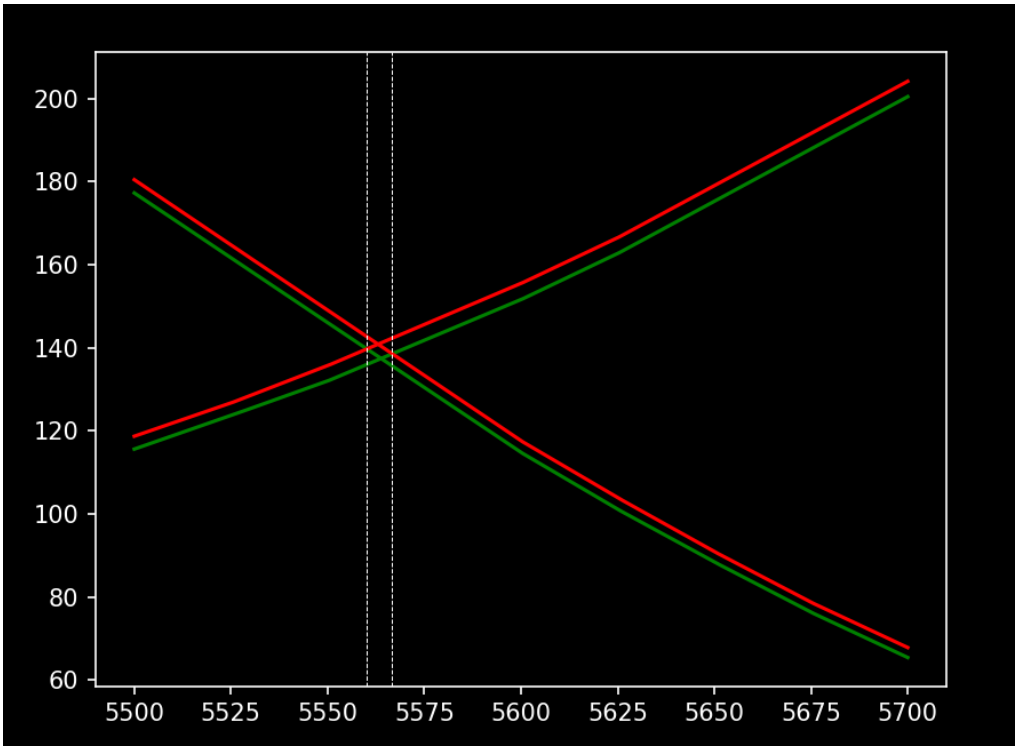
The price and implied volatility are then plotted against the strike or forward log-moneyness, $\ln\left(\frac{K}{Se^{\int_0^T R(t)dt}}\right)$.



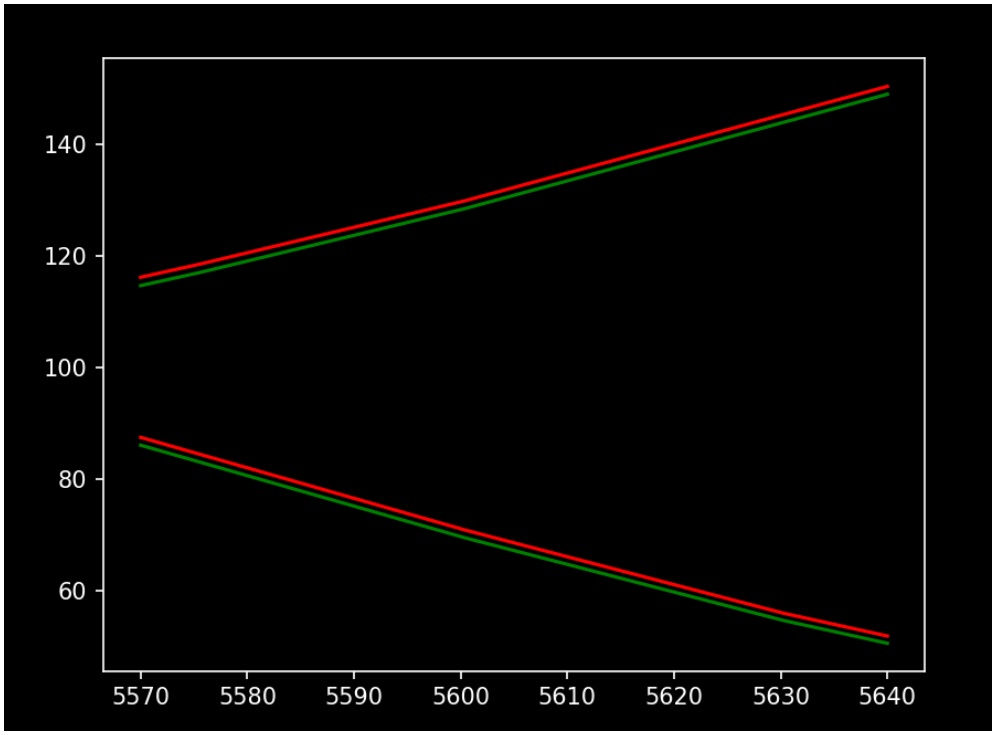


There are three issues with this. Firstly, there is a jump at the ATM forward strike, which is more obvious in the volatility space. This could be because we are using the wrong forward price, which is naively assumed to be the spot price multiplied by the risk-free growth rate. Secondly, yfinance combines weekly and monthly options expiring on the same day. However, weekly options are PM-settled (closing price), while monthly options are AM-settled (opening price). Lastly, there is noise in the option price data, which could result in a volatility surface that allows arbitrage.

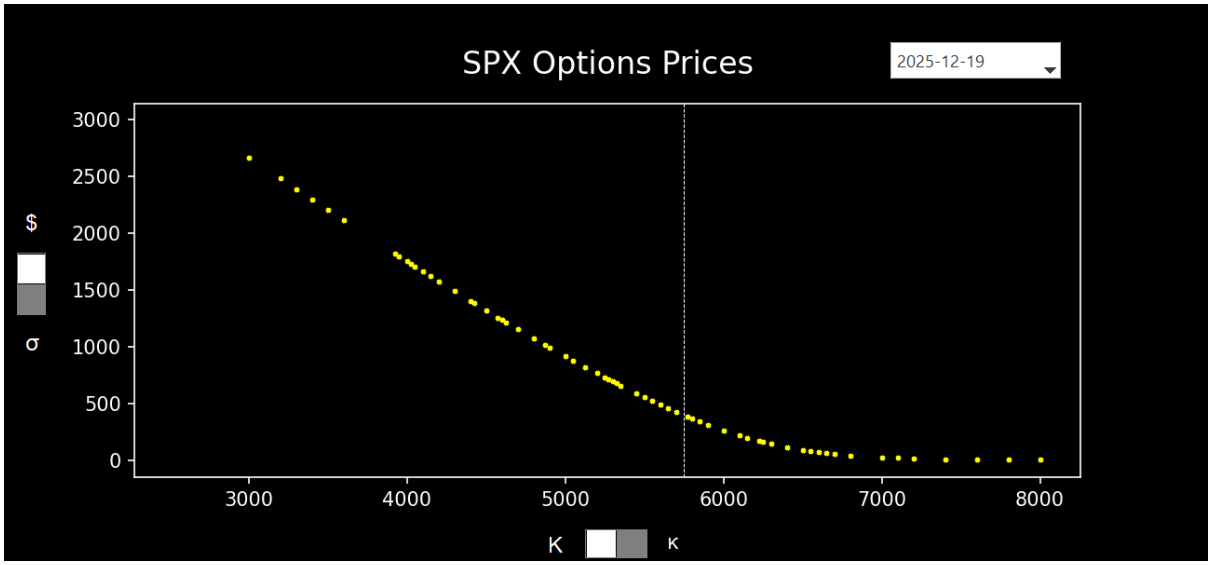
To solve the first problem, we note that in the Put-Call parity, when the strike is equal to the forward price, the call and put prices are identical. Rather than estimating the forward price using the risk-free rate, then filtering puts and calls based on this, we will estimate the forward price as the point at which the call and put price curves intersect each other.



In this graph, the red and green curves represent the ask and bid prices respectively, with the upward sloping curves being put prices and downward sloping curves being call prices. In this particular case, from this graph, the forward price is estimated to be between the two white dotted lines, 5560.10 and 5566.67. For most of the maturities, this interval is much narrower than 0.5%, which suggests that this estimate is quite accurate. We will take the mid-point of this interval to be the forward price.



If we do not have enough points to estimate the forward price for a particular maturity, as shown in the graph above, we will simply remove that maturity.



This results in price-strike and volatility-strike curves that are smooth at the ATM forward strike.

For the second problem, we separate the weekly options from the monthly options, with the time to maturity of weekly options being 6.5 hours longer (settled on 4pm closing price rather than 9:30am opening price).

For the last problem, to produce an arbitrage free implied volatility surface, we must first smooth the volatility smiles in a way that does not allow arbitrage. To do this, we use the method suggested by Fengler (2009), re-formulating the problem into a minimisation problem, minimising

$$\sum_{i=1}^{n} w_i\{y_i - g(K_i)\}^2 + \lambda \int_a^b \{g''(K)\}^2 dv$$

w_i: positive weights

$\{u_i, y_i\}$: points to be smoothed

g: natural cubic spline

λ: smoothness parameter

If $y = (w_1 y_1, \dots, w_n y_n, 0, \dots, 0)^T, x = (g^T, \gamma^T)^T, A = (Q, -R^T), B = \begin{pmatrix} W_n & 0 \\ 0 & \lambda R \end{pmatrix}$,

then it can be shown that this problem is equivalent to the quadratic minimisation problem

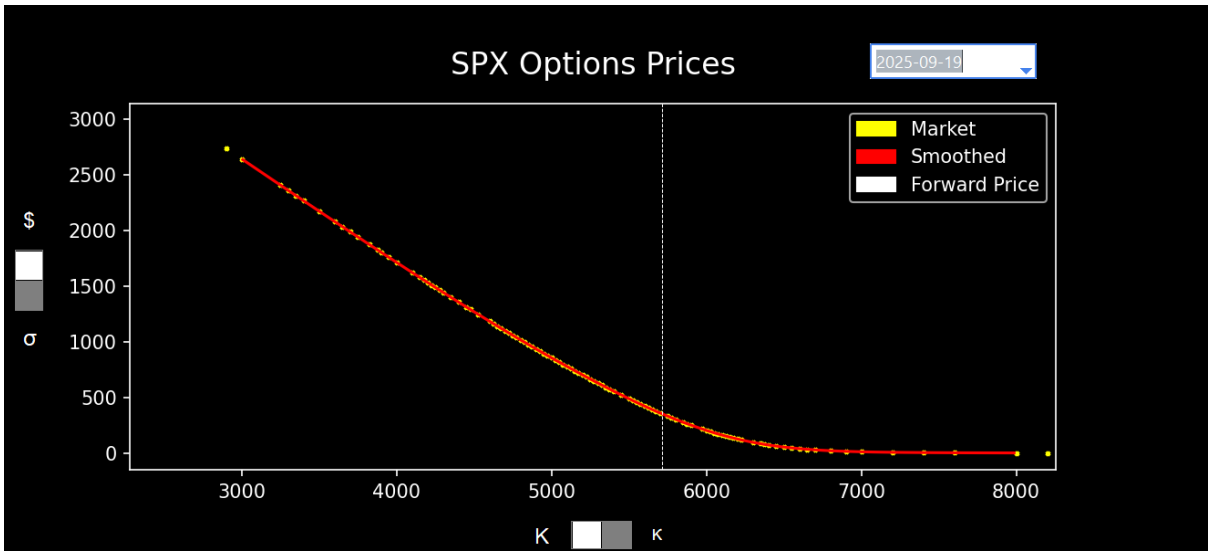
$\min -y^T x + \frac{1}{2} x^T B X$ subject to $A^T x = 0$

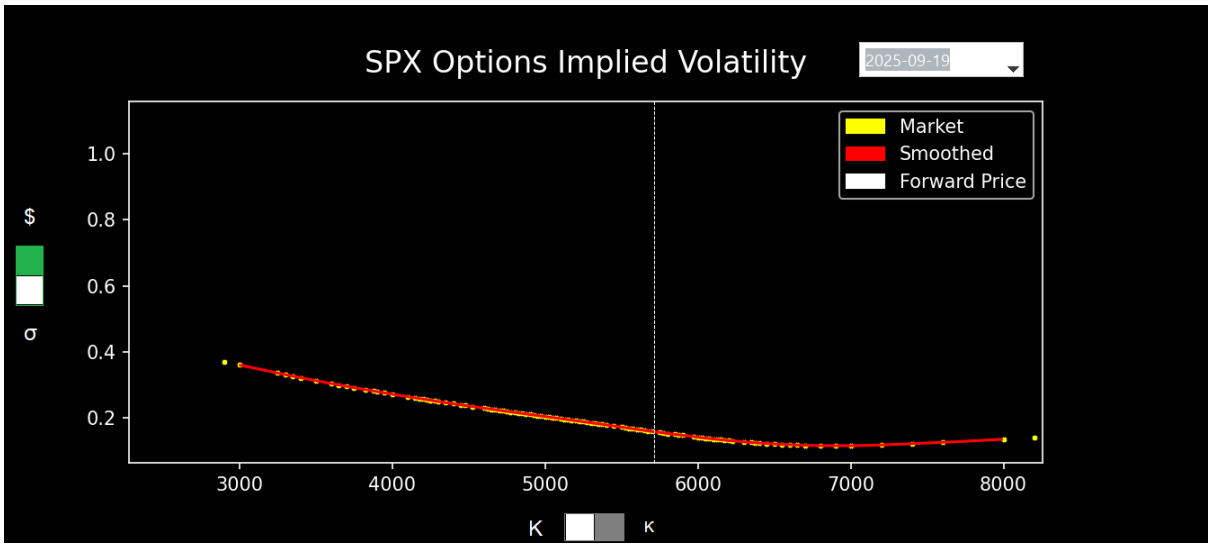
subject to additional constraints that ensures the convexity of price against strike and the standard bounds on call option prices.

We smooth the volatility smile for different maturities in reverse order, with the longest maturity first, while imposing additional constraints on subsequent minimisation

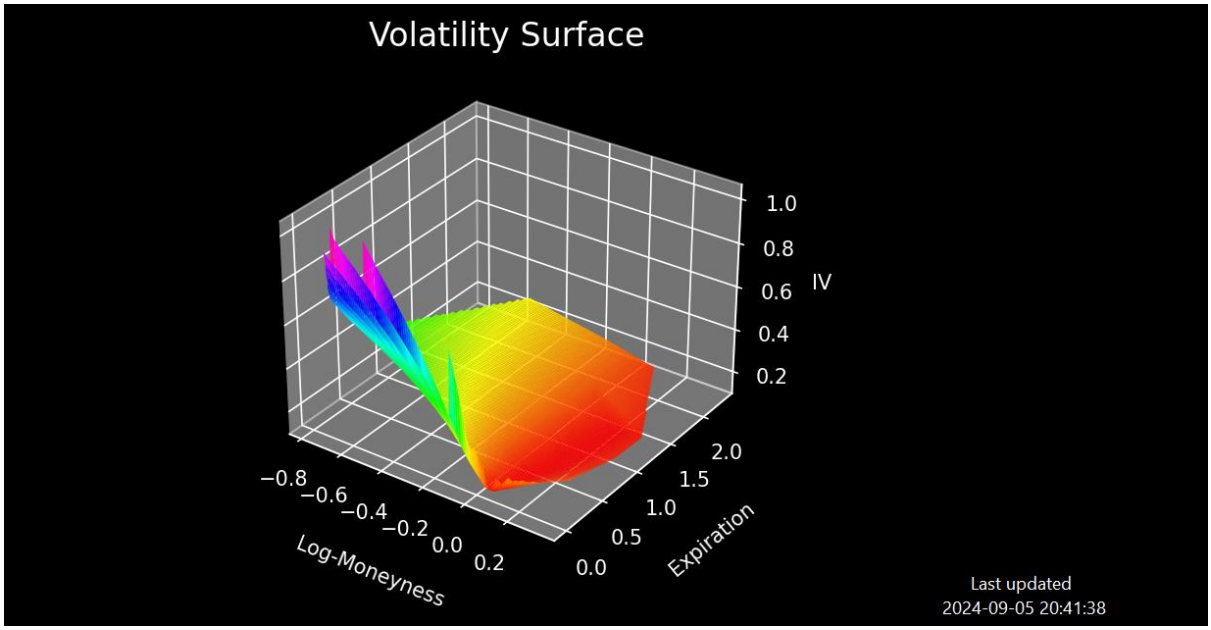
problems. This will preclude calendar arbitrage, i.e. options with different maturities but the same forward moneyness κ will have increasing prices with respect to maturity.

This is done using clarabel, an quadratic optimisation package in python, and gives the smooth price functions and volatility smiles shown below.





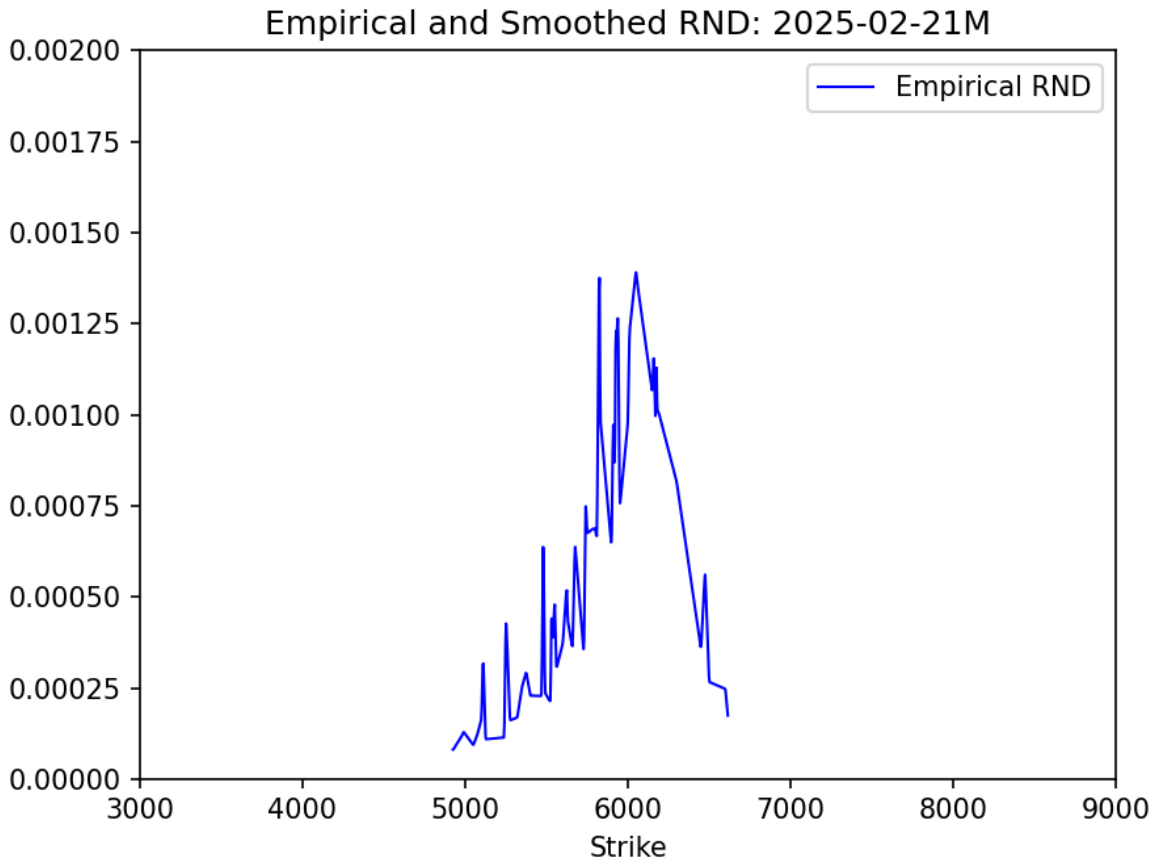
Then, using the griddata method in scipy for linear interpolation in the maturity axis, we obtain an arbitrage-free implied volatility surface that can be used for pricing.



To perform further analysis, we turn to the Breeden-Litzenberger formula to compute the empirical risk-neutral density:

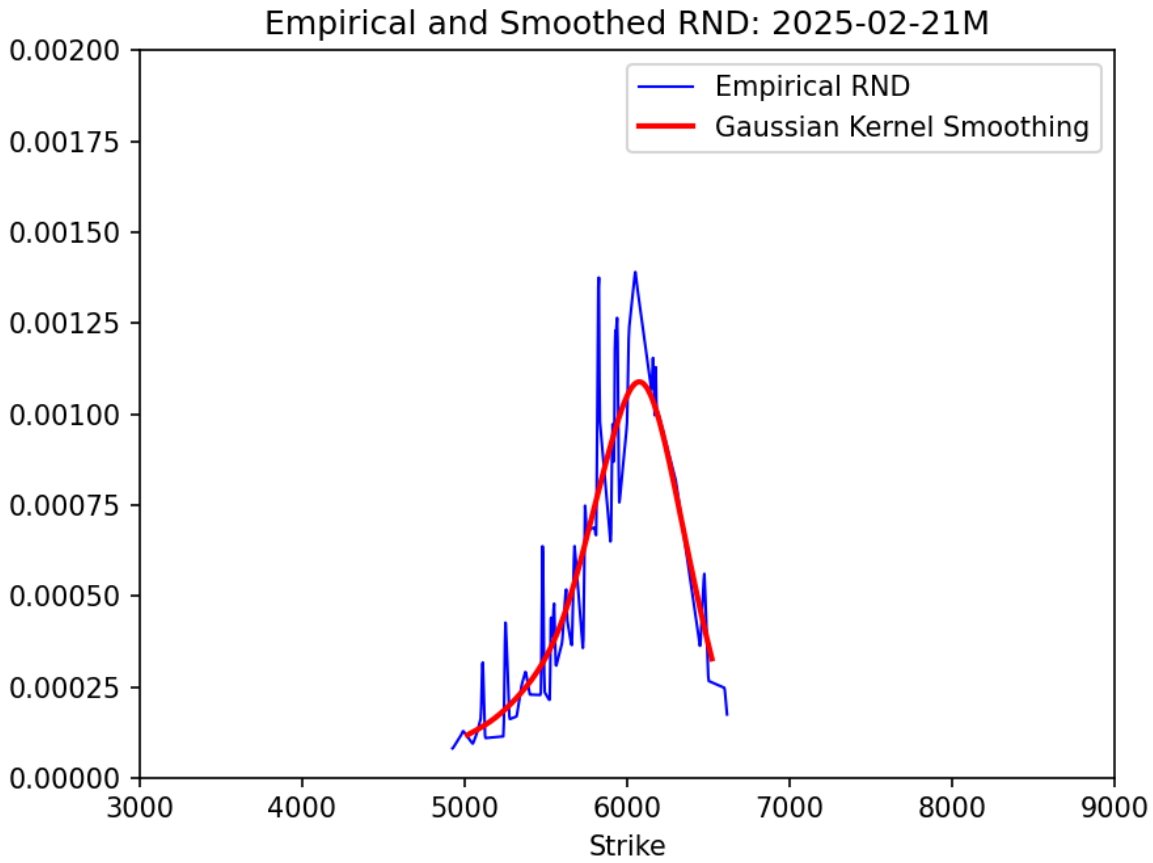
$$\frac{\partial^2 C(K,T)}{\partial K^2} = e^{rT} f_Q(K)$$

and in this case, using $\frac{estimated\ forward}{spot}$ as a proxy for e^{rT}.



Although the curves are smooth in the price and volatility space, the risk-neutral density obtained is very unsmooth, as we have not imposed any additional conditions on the second-order derivative of the smoothing spline other than continuity.

This is solved using the Gaussian filter function in SciPy, commonly used to blur images. The bandwidth/standard deviation for the Gaussian kernel is chosen qualitatively, large enough that it does not capture small fluctuations in the empirical density ($\sigma = 20$). The Gaussian filter produces the following result for the middle portion of the RND:



To extend the risk-neutral density to extreme values, the density of a generalised Pareto distribution – a distribution often used to model the tails of other distributions – is appended to the left and right tails. It has the following density function:

$$f(x;\sigma,\xi) = \frac{1}{\sigma}\left(1+\frac{\xi(x-\mu)}{\sigma}\right)^{\left(-\frac{1}{\xi}-1\right)}$$

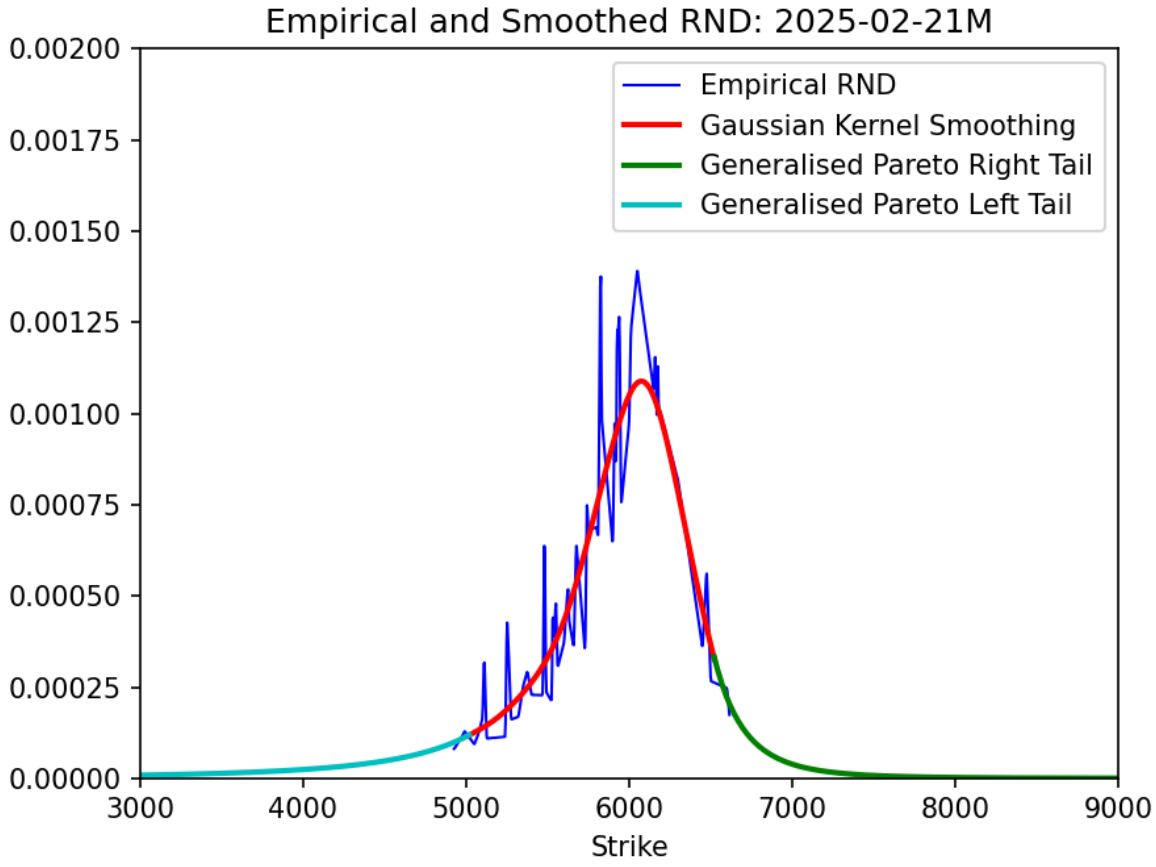
$$f'(x) = \frac{1}{\sigma^2}(-1-\xi)\left(1+\frac{\xi(x-\mu)}{\sigma}\right)^{\left(-\frac{1}{\xi}-2\right)}$$

This is done by matching the values and derivatives at the endpoints of the intervals, and setting μ to be the mode of the smoothed RND. Suppose the value and derivative at the right endpoint x are y and y' respectively , then we want

$$\frac{y'}{y} \approx \frac{f'(x)}{f(x)} = \frac{-1-\xi}{\sigma+\xi(x-\mu)}$$

$$i.e.\ \xi = -\frac{k\sigma + 1}{1 + k(x - \mu)}$$

Different values of σ are guessed, and the corresponding ξ values are calculated. The parameters are then chosen such that $f(x; \sigma, \xi)$ is sufficiently close to y. Finally, the curve is renormalised to ensure that the area under it equals to 1 approximately, which gives the following smooth RND extended to extreme values.



References:

Fengler, M. (2009). Arbitrage-free smoothing of the implied volatility surface

Green, P. J. and Silverman, B. W. (1994). Nonparametric Regression and Generalized Linear Models

Figlewski, S. (2012). Anatomy of a meltdown: The risk neutral density for the S&P 500 in the fall of 2008