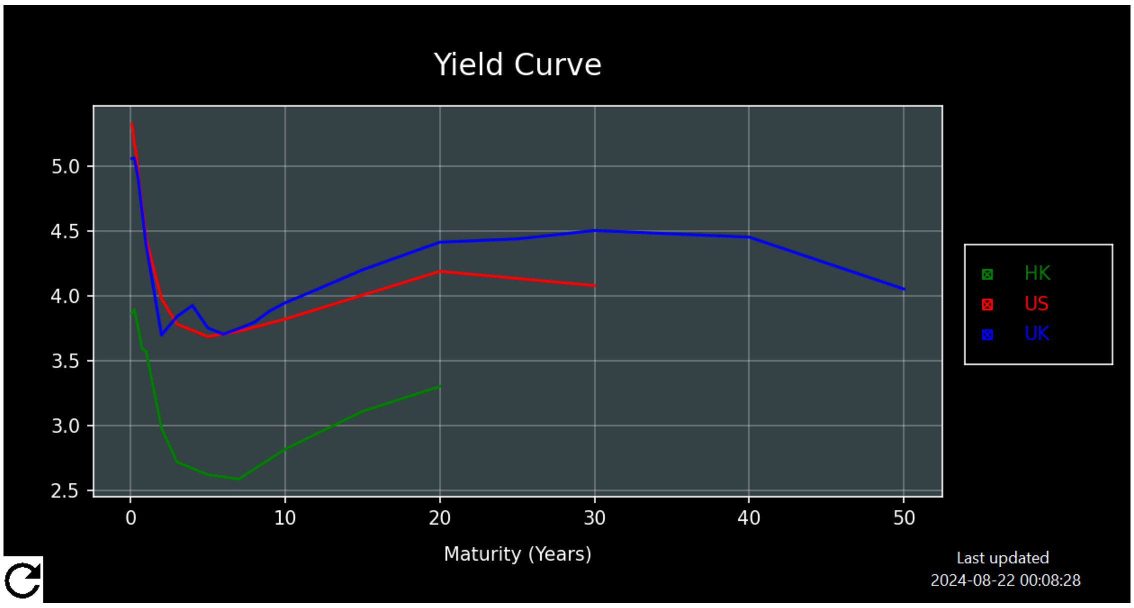
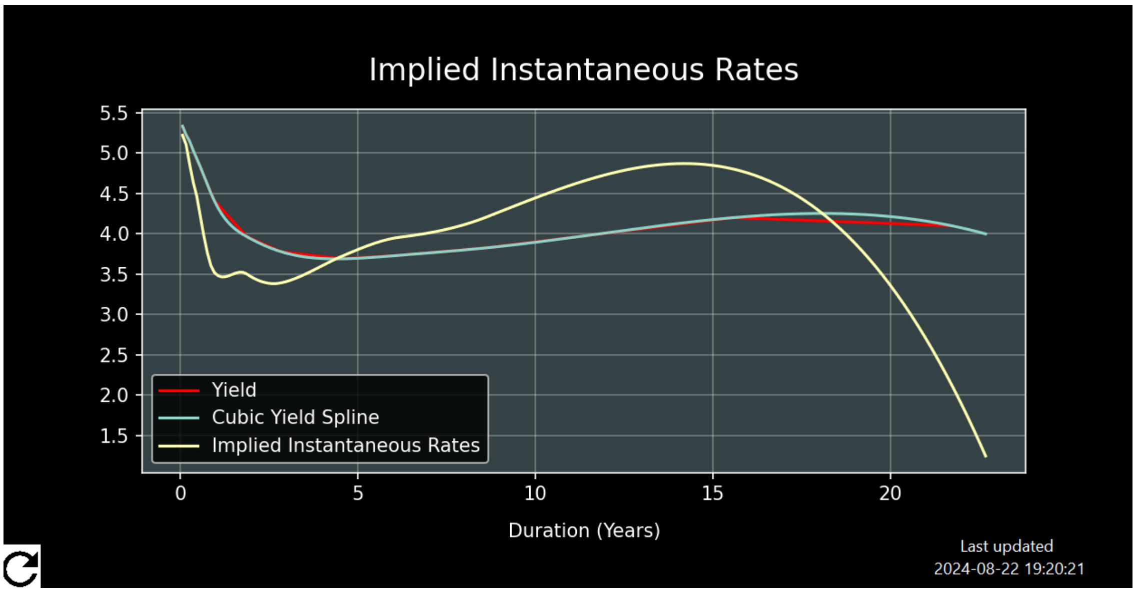
Volatility Surface

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 $t \in [0, T]$.

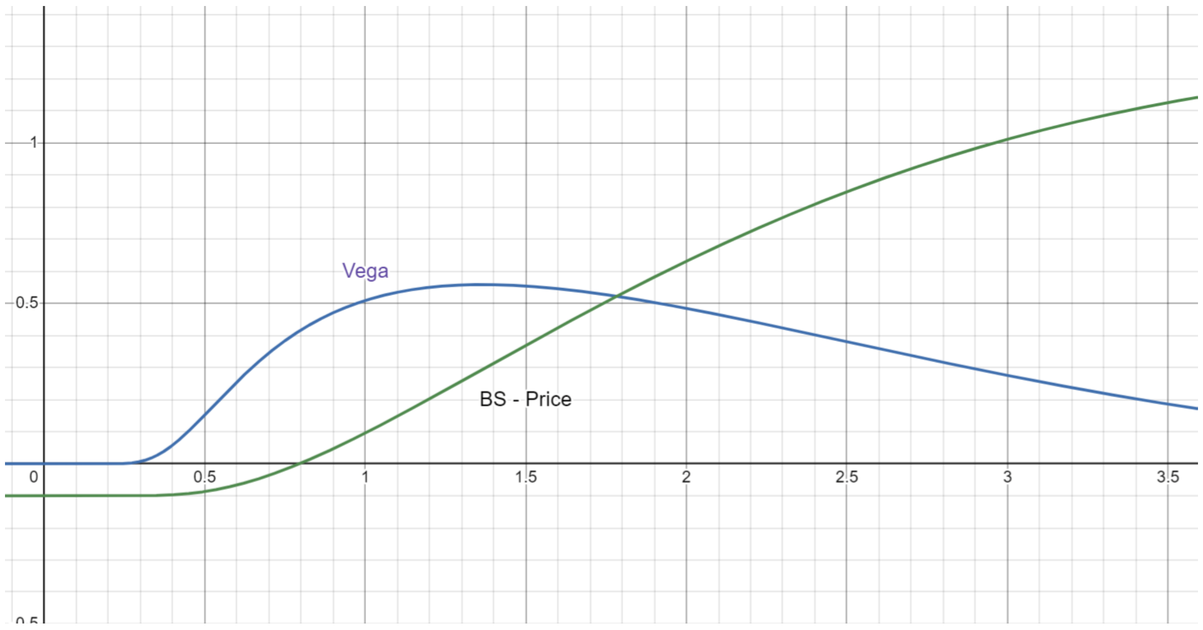
For each maturity, we use the Newton-Raphson numerical method to solve for the root σ that gives

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

with initial guess

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

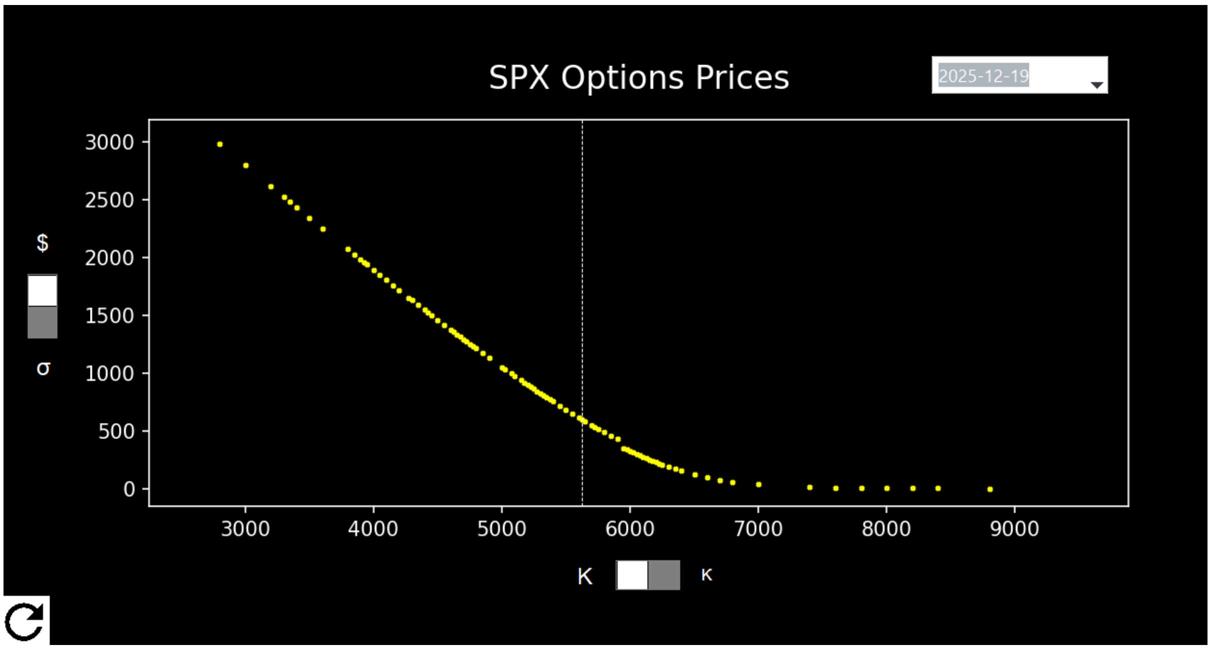
which is the inflection point on the graph of price against volatility and the maximum point on the graph of vega against volatility, to ensure that the answer converges. The target error in the algorithm is set to 10^{-5}, such that the process terminates when the estimated price and market price differ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

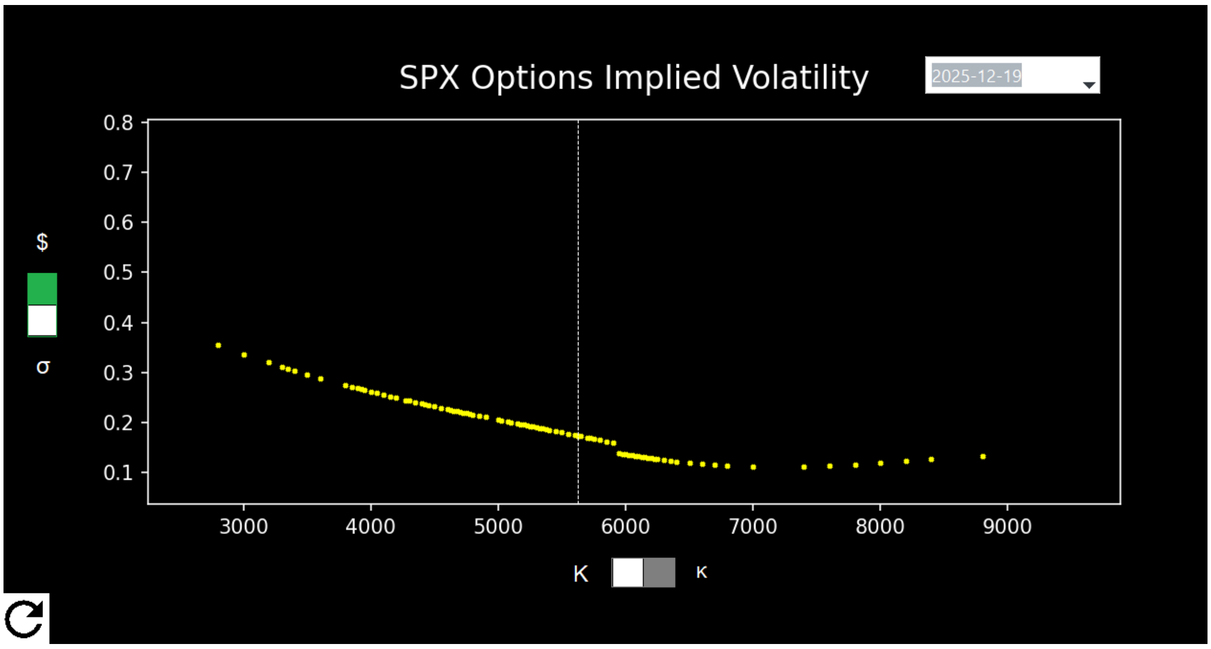


```python
def option_price_vega(option_type: str, S, K, T, r, sigma):
    d1 = math.log(S/K) / (sigma*math.sqrt(T)) + (r/sigma + sigma/2) * math.sqrt(T)
    d2 = d1 - sigma * math.sqrt(T)
    vega = S * math.sqrt(T) * norm.pdf(d1)
    if option_type == 'Call':
        price = S * norm.cdf(d1) - K * math.exp(-r * 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, $\ln\left(\frac{K}{Se^{\int_0^T R(t)dt}}\right)$.





We can see that there are some unsmooth points in both the price and implied volatility space. 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 $y = (w_1 y_1, \dots, w_n y_n, 0, \dots, 0)^T, x = (g^T, \gamma^T)^T, A = (Q, -R^T), B = \begin{pmatrix} W_n & 0 \\ 0 & \lambda R \end{pmatrix}$,

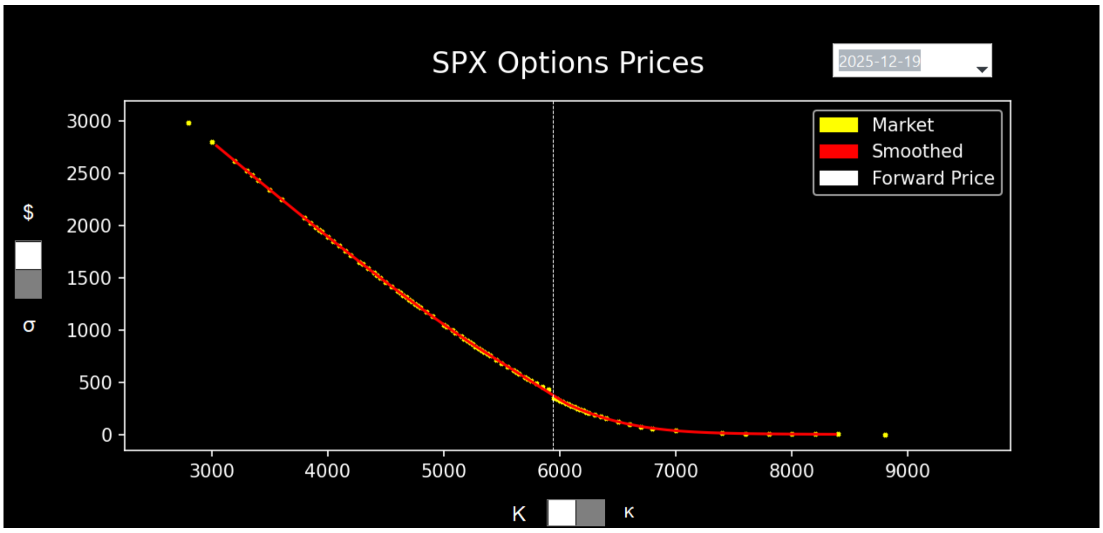
then it can be shown that this problem is equivalent to the quadratic minimisation problem

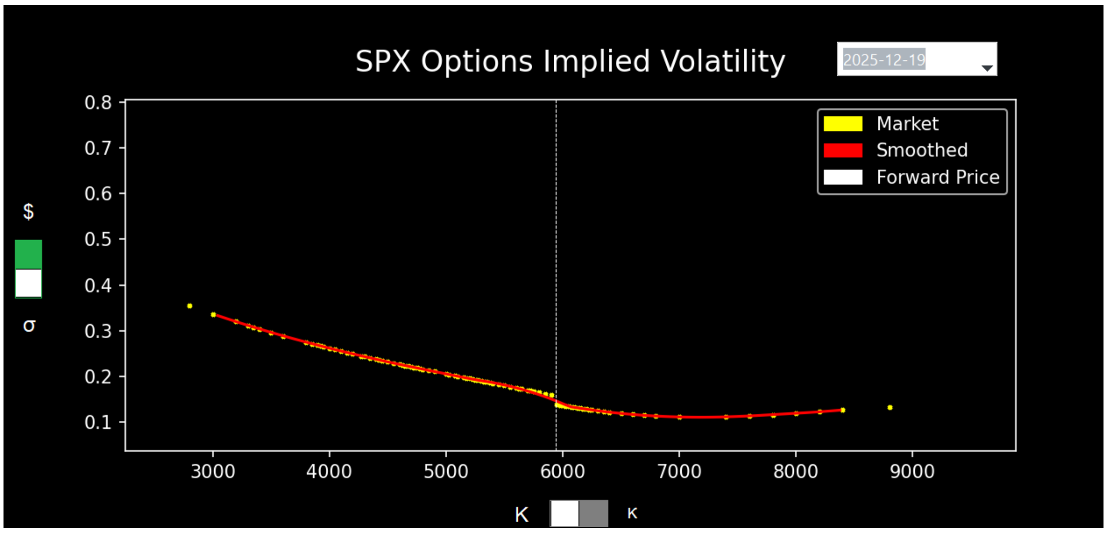
$$\min -y^T x + \frac{1}{2} x^T B X \text{ subject to } A^T x = 0$$

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

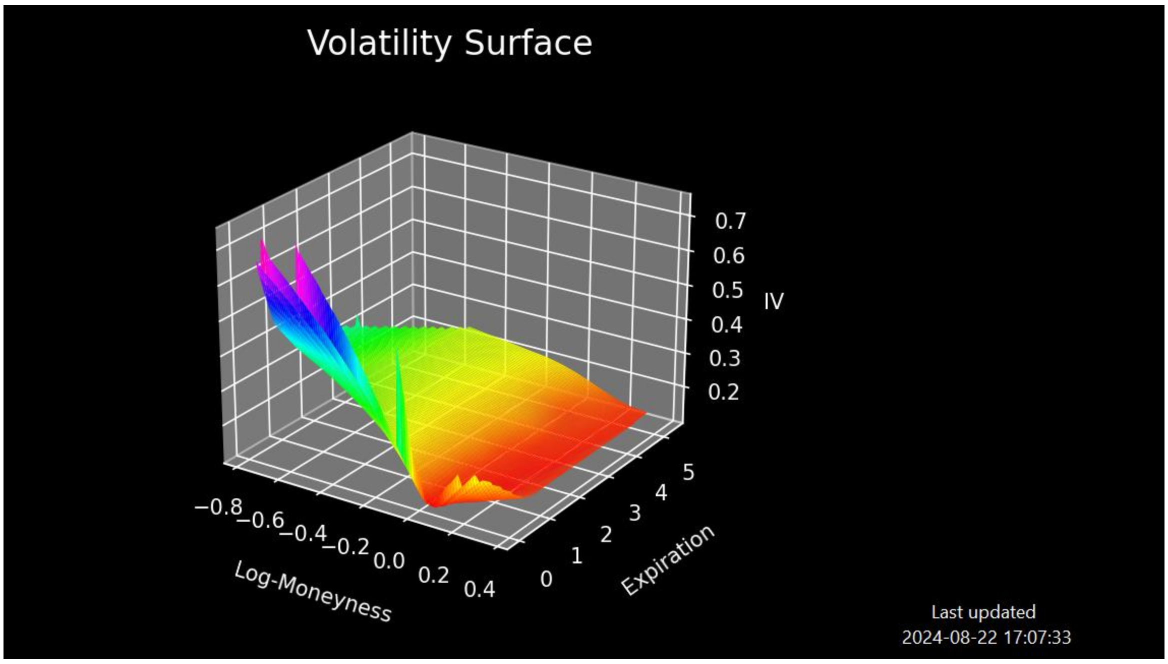
We smooth the volatility smile for different maturities in reverse order, with the longest maturity first, while imposing additional constraints on subsequent minimisation problems. This will preclude calendar arbitrage, i.e. options with different maturities but the same forward moneyness will have increasing prices with respect to maturity.

This is done using clarabel, an 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.



References:

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

Green, P. J. and Silverman, B. W. (1994). Nonparametric