Contents

Model selection

To create a volatility surface, one approach is to interpolate our discrete data. This allows for a continuous surface of implied volatility across all strikes and maturities. From researching, I discovered that a common interpolation technique used for volatility surfaces is 'cubic spline', which I have implemented for the data. However, after further researching, I found that more recent approaches have used parametric models instead, typically with OLS regression and use of PCA's. After reading more literature about this area and looking at already existing models, I decided to test and compare the following models below, where I first explore the parametric models before moving on to the interpolation method.

Model 1 (Benchmark): $IV = B_0 + B_1K + B_2K^2$

Model 2: $IV = B_0 + B_1K + B_2K^2 + B_3T + B_4T^2$

Model 3: $IV = B_0 + B_1K + B_2K^2 + B_3T + B_4T^2 + B_5(K \cdot T)$

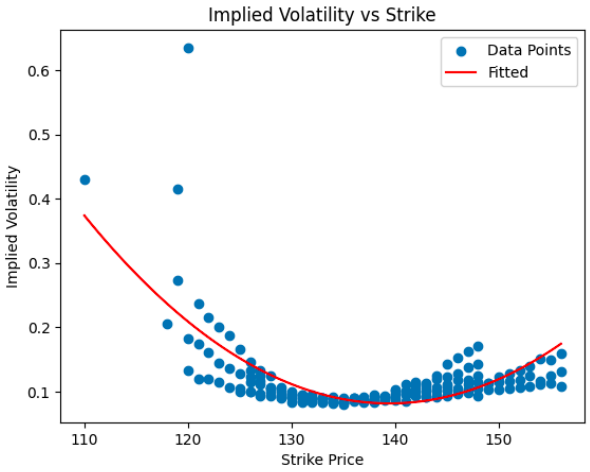
Model 4: $IV = B_0 + B_1K + B_2K^2 + B_3T + B_4T^2 + B_5(K \cdot T) + B_6K^3 + B_7K^4$

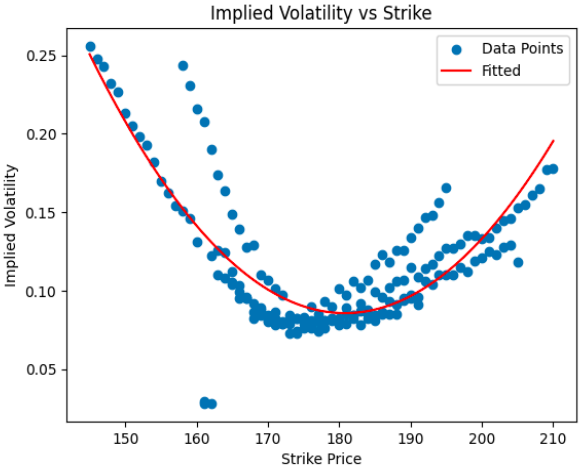
Model 5: Cubic Spline Interpolation

Where IV = Implied Volatility, K = Strike, T = Expiry (Maturity) and the Betas are the corresponding regression coefficients.

The first model is a simple polynomial which will be used as the benchmark, the choice of this will be explained later in this document. The second model additionally captures the quadratic effect of the expiry and the final model further captures the interaction term. I have also noticed that some models use Moneyness rather than Strike price, however I will not use that as a parameter here.

Model 1 (benchmark):



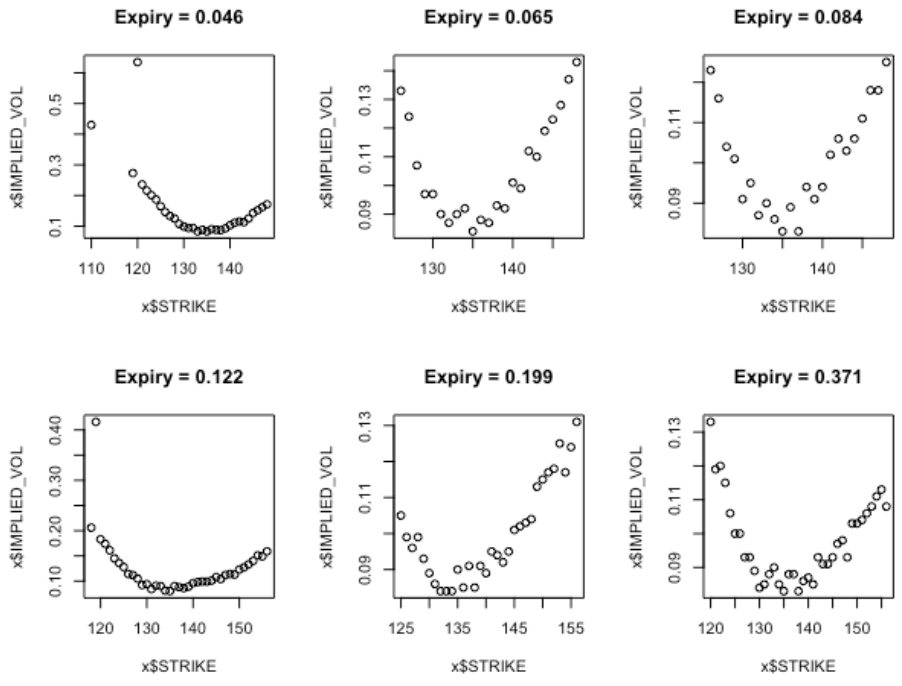


Parameters	Values (rounded)	Significant Level	
B_0	6.65	***	
B_1	-0.095	***	
B_2	0.00034	***	$R^2_A = 0.47$ $R^2_B = 0.66$

(Significant code: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1)

The first two plots are the fitted smiles regardless of option maturity, however, as mentioned in the question I will also provide a smile for each maturity. However, I do not use this approach when fitting my surface as the other models provide better results.

Plot of smiles for each maturity (dataset 1):



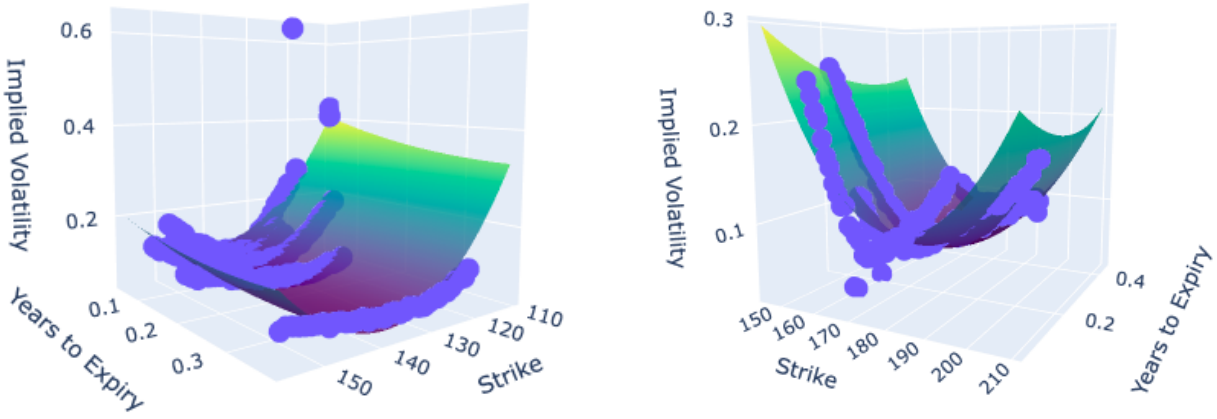
Please note I will only include the R-squared value for the model on the second dataset and not the coefficients for better readability.

As expected the volatility surface is just a simple two dimensional plot of a volatility smile. The simple polynomial seems like a good start to model volatility as from the plot we can see a clear non-linear trend. The R-squared value for dataset 1 is 0.47 and 0.66 for dataset 2, simply put, this means 47% / 66% of the variation in implied volatility can be explained by the strike price.

Also worth noting is that implied volatility is lowest when the strike price is close to the underlying asset creating the 'smile' effect. Researching this, one hypothesis is that the demand is greater for options in-the-money or out-the-money rather than options at-the-money. We will see this effect across all models, therefore I will not mention it for every model specifically.

Using the API function I created (code can be found in Appendix A) to retrieve implied volatility. For dataset 1, a strike price of 126 and expiry 0.122 the implied volatility is 0.142. These numbers were only chosen for comparison reasons, where 126 is the most common strike and 0.112 is the most common expiry.

Model 2:



Please note: The axis orientation on the second plot is different to make the plot clearer.

Parameters	Values (rounded)	Significant Level	
B_0	7.03	***	
B_1	-0.095	***	
B_2	0.00036	***	

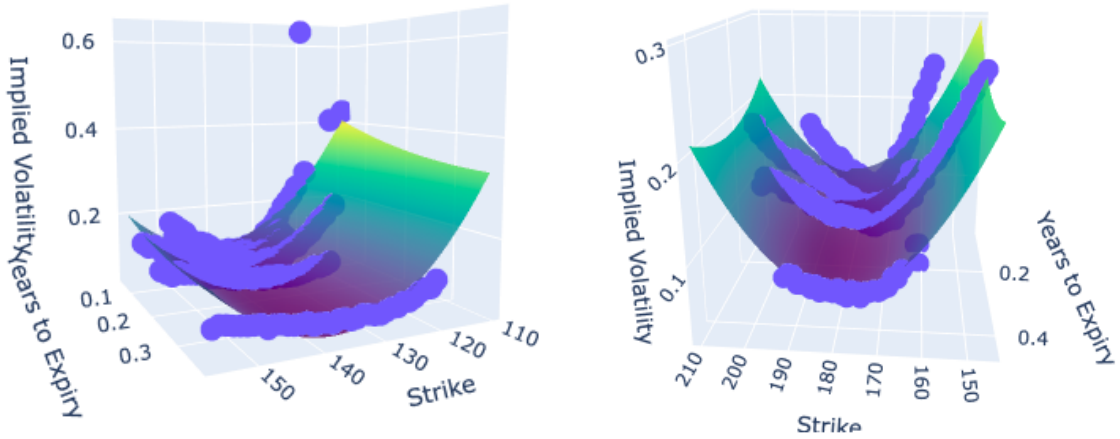
B_3	-0.45	***	
B_4	0.72	*	$R^2_A = 0.55$ $R^2_B = 0.77$

(Significant code: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1)

Straight away we see an improvement in the goodness of fit measure, R-squared, which is now 55% and 77%. This clearly shows that expiry and its quadratic effect does help to model implied volatility. Although the quadratic effect of expiry has a higher p-value, it is still statistically significant at the 95% level, therefore we still include this in our model. We can also see that the model does not overfit to the outliers in dataset 1.

Again using the API for a strike of 126 and expiry 0.122 the implied volatility using model 2 is 0.138. Suggesting that model 1 overestimates the IV.

Model 3:



Parameters	Values (rounded)	Significant Level	
B_0	6.91	***	
B_1	-0.097	***	
B_2	0.00035	***	
B_3	-0.88	**	
B_4	0.67	***	

B_5	0.0033	–	$R^2_A = 0.56$ $R^2_B = 0.78$

(Significant code: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '_' 1)

This table is quite interesting as we see that apparently the interaction term is not significant for this data. Having researched many parametric models for volatility, I found they all contained an interaction term, therefore it was interesting that for this data, the term was not significant.

I have included an ANOVA table in Appendix B to show the interaction is not needed.

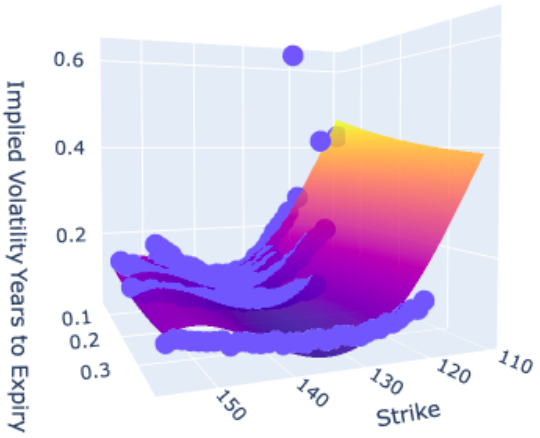
The R-squared value only increases by 0.1 for both datasets, further suggesting that the interaction term is not needed. Therefore I would choose model 2 when modelling implied volatility.

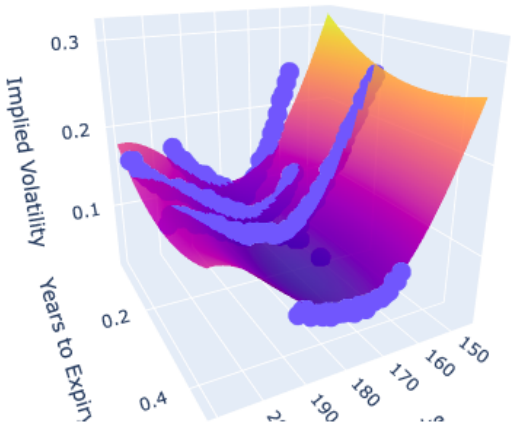
Using the API for strike 126 and expiry 0.122 gives an IV of 0.140.

Having researched other models, some included terms with very high degrees and utilised principal components. However, given the simplicity of the dataset, I think model 2 is an okay non-complex model to estimate IV, however, ideally I would try to build a model with a much higher R-squared using more complex methods.

When playing around with more complex models, including significant terms up to degree 4, the highest R-squared I could achieve was still only 0.66 which is much better but more complex.

Model 4 (Over-complex):





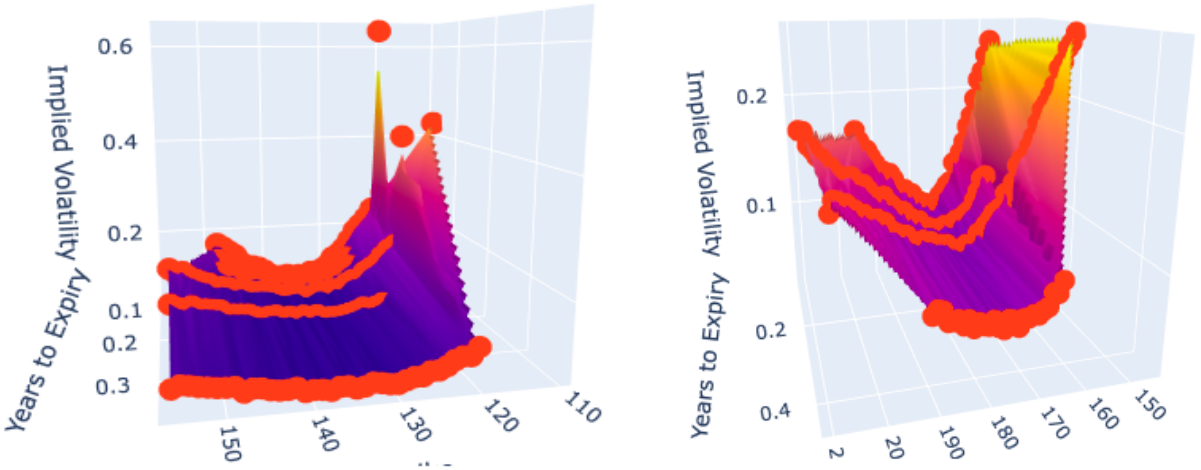
Straight away, we can see this more complex model fits the data much better, being able to identify curves in the data model 2 and 3 could not find. However, this could be overfitting, finding patterns only present in the given dataset and will not be applicable in new (real) data.

What's also interesting is that the interaction term is now significant in the presence of the terms to the power of 3 & 4. This model is a lot more complex and less computationally efficient compared to model 2, and can be very prone to overfitting the training data used to fit the model.

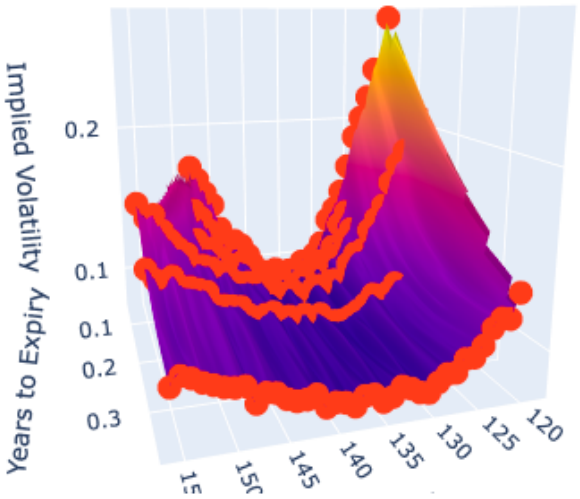
Worth noting: The AIC for this model was the smallest compared to all the others, implying that by this criterion, model 4 provides the best balance of goodness of fit and complexity.

Model 5 - Cubic Spline Interpolation

Now moving away from the parametric models, I used interpolation to create a volatility surface testing both the linear and cubic spline method.



Straight away, it's clear that the interpolation approach struggles with outliers, too sensitive. Therefore, I decided to remove the outliers and re-fit using this approach. I identified the outliers through residual analysis. The elements removed in dataset 1 were [0,3,4].



We can now see the surface fits the data much smoother. One issue I came across was choosing the number of points in the meshgrid (see code in Appendix A). Choosing a lower number would result in a smoother surface but could miss variations in the data whilst choosing a higher number would lead to overfitting.

One advantage of using an interpolation approach is its relatively simple (essentially just filling in the gaps) and is computationally more efficient than a regression approach.
However, interpolation is very prone to overfitting and is very sensitive to outliers.

Arbitrage and Financial Implication of Implied Volatility

To start with a fundamental principle, both call and put options have a higher price if implied volatility is higher. Therefore when pricing options and finding mispricing, implied volatility is extremely important.
Now, also fundamental, implied volatility is essentially the value of volatility used in the Black-Scholes formula to achieve the market option price. This can be found using various numerical methods, the most popular I found when researching is the Newton-Raphson method due to its fast convergence rate.
The benefit of doing this is that we can compare this implied volatility with our surface and try to capitalise on any mispricing.

A simple scenario:
If we find an option on the market with an implied volatility lower than the implied volatility from our model. We would buy this option as we believe it's priced too low and delta-hedge accordingly by trading the underlying asset. To explain simply, delta hedging is buying or selling a certain amount of the underlying asset, determined by the options delta, to offset the risk. I have only studied delta hedging using the binomial model, where it's fairly simple to obtain the delta values given probabilities of the assets movement. In reality, a more complicated model would be needed.

Focussing on the dataset:
From the plots, it's clear to see there are outliers present which we can use to find an arbitrage opportunity. From dataset 1: an obvious outlier is the option with strike 120 and expiry 0.046 has implied volatility 0.635. From our fitted surface, we can clearly see that this implied volatility is far too high. Therefore a strategy would be to short this option as it is priced too high.

Now for dataset 2: There are 2 options with strike 161 that have implied volatility too low (0.028 & 0.029). Therefore a strategy would be to buy these options as the low IV suggests the option is priced too low.

There are other ways to try to find arbitrage using this data. If we find options with the same strike but with different expiries we can use a calendar spread to try and benefit from this. When researching I found conditions that volatility surfaces have to satisfy to be arbitrage-free:

Arbitrage check

A grid for the market price data in (K,T) is set up and the strikes are denoted $K_1, K_2, K_3, ..., K_{kmax}$ and the maturities $T_1, T_2, T_3, ..., T_{tmax}$, both in ascending order. Next the option prices are looped through using $i = 1, 2, 3, ..., kmax$ and $j = 1, 2, 3, ..., tmax$ and the following tests are made at each point:

1.Positivity of the calendar spread: $C(K_i, T_{j+1}) - C(K_i, T_j) > 0$.

2.Positivity of the vertical spread: $C(K_i, T_j) - C(K_{i+1}, T_j) > 0$.

3.Positivity of the butterfly spread: $C(K_i, T_j) - \frac{K_{i+2} - K_i}{K_{i+1} - K_i} C(K_{i+1}, T_j) + C(K_{i+2}, T_j) > 0$

4.Bounds for the price, (A4): $max(S_0 - K_i, 0) \leq C(K_i, T_j) \leq S_0$

If any of the conditions are not fulfilled for a data point, the lowest price offset in percent could be printed out together with the current (K,T) and a boolean for arbitrage could be returned as false.

Unfortunately, we do not have access to the price of the options therefore we can not check if these conditions hold. However, if we assume a fixed interest rate and other necessary conditions to satisfy Black-Scholes, we would be able to estimate the option prices and check these conditions.

I also found these conditions to check for butterfly arbitrage:

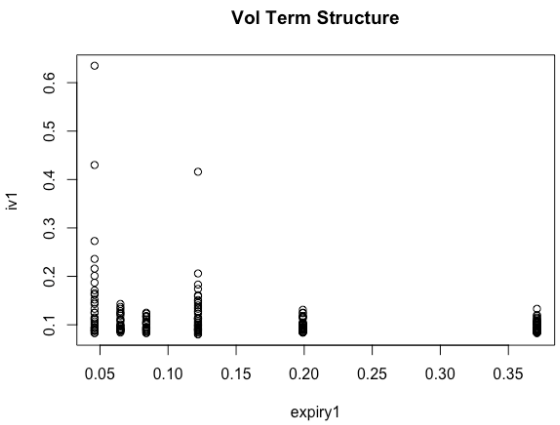
Theorem 4.2

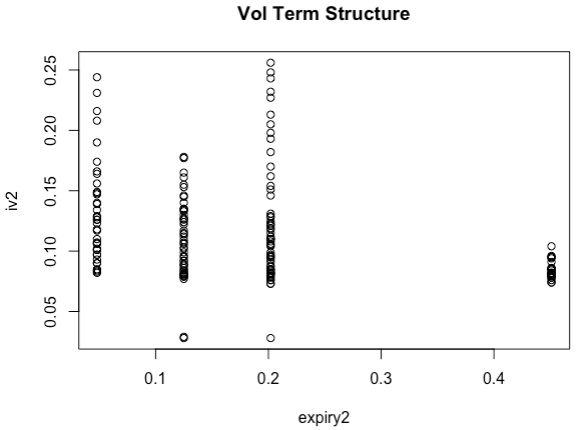
The volatility surface (1) *is free of butterfly arbitrage if the following conditions are satisfied for all* $\theta > 0$*:*

1. $\theta\varphi(\theta)(1 + |\rho|) < 4$;
2. $\theta\varphi(\theta)^2(1 + |\rho|) \leq 4$.

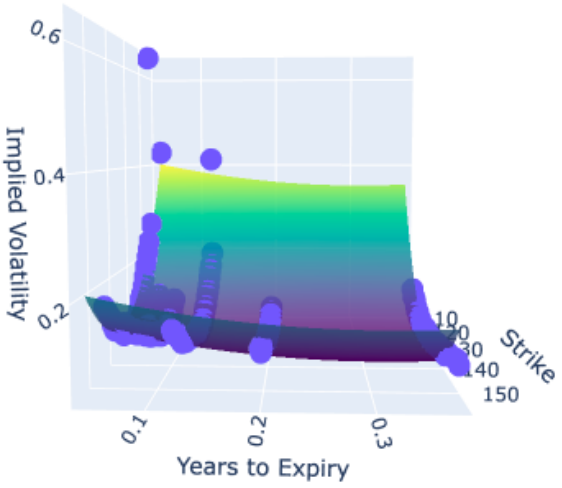
As these conditions are not challenging to code, I was able to test these conditions on our dataset (see code in Appendix A). The output suggests that these conditions are violated and arbitrage may exist. Researching this further, one hypothesis states that this issue arises when the surface is too steep in some places.

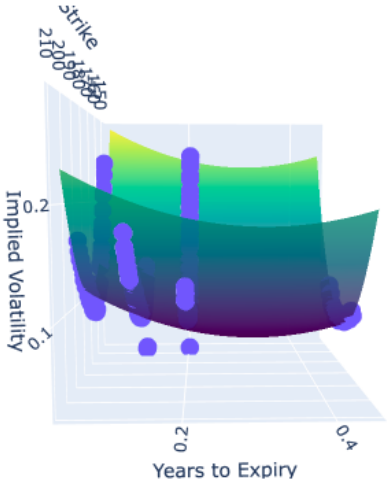
With calendar spread, it is useful to plot the volatility term structures:





From these plots, it's clear to see that for the datasets given, implied volatility is higher for options with a shorter time to expiry. This explains why the surface has a slight slope down, shown clearly in the next plots.



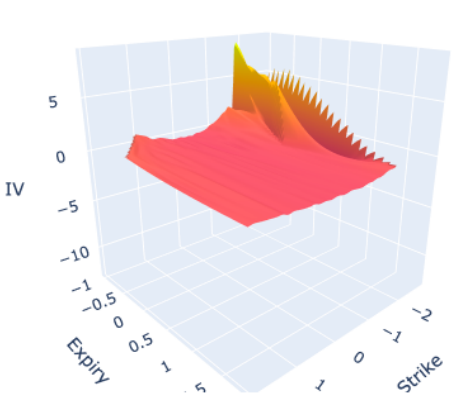


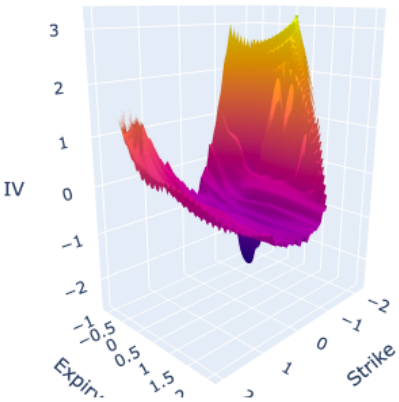
Other approaches I tried

Here are some other approaches I used to attempt to fit the volatility surface:

I tried fitting this surface using PCA's, however, my results were not great.

PCA model:





Whilst the axis may look incorrect, this is because I have standardised the data first as PCA's are very sensitive to the scale of variables and will put greater weight on variables with larger numbers.

The surfaces do not seem to be a great fit and seem to be greatly affected by the outliers in the dataset. An approach to solve this issue would be to simply remove these options, however, this would lead to bias.

The R-squared for this model is 0.252 which is very low, further sporting that this model does not fit well.

Further researching this, studies have shown that using an Arrow-Debreu state space approach can solve these issues, however, unfortunately I could not find anything concrete and worth testing.

SVI model:

I tried to fit an SVI based on the information I found in the Baruch College (University of New York) lecture notes that were available online.

However, when I tried to plot this volatility surface, the surface was only linear and seemed to have no quadratic effect. This is something I would have liked to investigate further as I assume my code is incorrect somewhere.

Appendix A - Python code

Please note: I have loaded the datasets in from Google Drive and called them data1 & data2 respectively. The initial code may have to be modified on your Python notebook.

Load in Data

```
import pandas as pd
drive.mount('/content/drive')
data1 = pd.read_csv('/content/drive/My Drive/synthetic-dataset1.csv')
data2 = pd.read_csv('/content/drive/My Drive/synthetic-dataset2.csv')
```

Initial Simple Polynomial fit

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

def polynomial_fit(x, a, b, c):
   return a * x**2 + b * x + c

def fit_surface(data):
   coef, _ = curve_fit(polynomial_fit, data['STRIKE'],
data['IMPLIED_VOL'])
   return coef

coeffs1 = fit_surface(data1)
coeffs2 = fit_surface(data2)

def plot_surface(data, coeffs):
   plt.scatter(data['STRIKE'], data['IMPLIED_VOL'], label='Data
Points')
   x_values = np.linspace(min(data['STRIKE']), max(data['STRIKE']),
100)
   y_values = polynomial_fit(x_values, *coeffs)
   plt.plot(x_values, y_values, color='red', label='Fitted')
   plt.xlabel('Strike Price')
   plt.ylabel('Implied Volatility')
   plt.title('Implied Volatility vs Strike')
   plt.legend()
   plt.show()

plot_surface(data1, coeffs1)
plot_surface(data2, coeffs2)
```

Create a Vol surface where I change poly(...) and surface function(...) based on each model. Produces 3D interactive plot

```
import plotly.graph_objects as go

def surface_fit(data):
    def poly(xy, a, b, c, d, e, f):
        x, y = xy[:, 0], xy[:, 1]
        return a * x**2 + b * y**2 + c * x * y + d * x + e * y + f

    xy_data = np.column_stack((data['STRIKE'], data['YEARS_TO_EXPIRY']))
    z_data = data['IMPLIED_VOL']

    coef, _ = curve_fit(poly, xy_data, z_data)
    return coef

surface_coeffs1 = surface_fit(data1)
surface_coeffs2 = surface_fit(data2)

def surface_function(xy, a, b, c, d, e, f):
    x, y = xy
    return a * x**2 + b * y**2 + c * x * y + d * x + e * y + f

def plot_surface(data, coeffs):
    x_values = np.linspace(min(data['STRIKE']), max(data['STRIKE']),
100)
    y_values = np.linspace(min(data['YEARS_TO_EXPIRY']),
max(data['YEARS_TO_EXPIRY']), 100)
    x_mesh, y_mesh = np.meshgrid(x_values, y_values)

    z_values = surface_function((x_mesh, y_mesh), *coeffs)

    fig = go.Figure(data=[
        go.Scatter3d(x=data['STRIKE'], y=data['YEARS_TO_EXPIRY'],
z=data['IMPLIED_VOL'], mode='markers', name='Data Points'),
        go.Surface(x=x_mesh, y=y_mesh, z=z_values, opacity=0.8,
colorscale='plasma', name='Fitted Surface')
    ])
    fig.update_layout(scene=dict(
        xaxis_title='Strike',
        yaxis_title='Years to Expiry',
        zaxis_title='Implied Volatility'
    ))
```

```
    fig.show()


plot_surface(data1, surface_coeffs1)

plot_surface(data2, surface_coeffs2)
```

API to retrieve Implied Vol for a given option

```
def get_implied_volatility(strike, years_to_expiry, coeffs):
    implied_vol = surface_function((strike, years_to_expiry), *coeffs)
    return implied_vol

strike = 126
years_to_expiry = 0.122
implied_volatility = get_implied_volatility(strike, years_to_expiry,
surface_coeffs1)
print("Implied Volatility:", implied_volatility)
```

Checking Theorem 4.2

```
theta_values = data1['YEARS_TO_EXPIRY']
vol_skew_values = data1['IMPLIED_VOL']

vol_skew_derivative = np.gradient(vol_skew_values, theta_values)

condition_1 = theta_values * vol_skew_derivative * (1 + np.abs(0.5)) <
4
condition_2 = 0.5 * theta_values * vol_skew_derivative * (1 +
np.abs(0.5)) <= 4

if np.all(condition_1) and np.all(condition_2):
    print("The volatility surface is free of butterfly arbitrage.")
else:
    print("The volatility surface may contain butterfly arbitrage.")
```

Initial Interpolation (linear)

```
import numpy as np
from scipy.interpolate import griddata

strike = data1['STRIKE']
```

```
mat = data1['YEARS_TO_EXPIRY'].values
iv = data1['IMPLIED_VOL'].values

s, m = np.meshgrid(np.linspace(min(strike), max(strike), 100),
np.linspace(min(mat), max(mat), 100))

interpol = griddata((strike, mat), iv, (s, m), method='linear')

fig = go.Figure(data=[go.Surface(z=interpol, x=s, y=m)])
fig.update_layout(title='Interpolated Surface
Plot',scene=dict(xaxis_title='Strike',yaxis_title='Years to
Expiry',zaxis_title='Implied Volatility'))

fig.add_scatter3d(x=data1['STRIKE'], y=data1['YEARS_TO_EXPIRY'],
z=data1['IMPLIED_VOL'], mode='markers')

fig.show()
```

Cubic Spline Interpolation without Outliers

```
import numpy as np
from scipy.interpolate import griddata
import plotly.graph_objects as go

remove = [0, 3, 4]

mask = np.ones(len(strike), dtype=bool)
mask[remove] = False

filtered_strike = data1['STRIKE'][mask]
filtered_iv = data1['IMPLIED_VOL'][mask]
filtered_mat = data1['YEARS_TO_EXPIRY'][mask]

s2, m2 = np.meshgrid(np.linspace(min(filtered_strike),
max(filtered_strike), 25),
                     np.linspace(min(filtered_mat), max(filtered_mat),
25))

interpol = griddata((filtered_strike, filtered_mat), filtered_iv, (s2,
m2), method='cubic')

fig = go.Figure(data=[go.Surface(z=interpol, x=s2, y=m2)])
```

```
fig.update_layout(title='Interpolated Surface
Plot',scene=dict(xaxis_title='Strike',yaxis_title='Years to
Expiry',zaxis_title='Implied Volatility'))


fig.add_scatter3d(x=filtered_strike, y=filtered_mat, z=filtered_iv,
mode='markers')

fig.show()
```

Appendix B - R screenshots

Model 1:

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.648e+00  5.491e-01   12.11   <2e-16 ***
strike1     -9.419e-02  8.044e-03  -11.71   <2e-16 ***
I(strike1^2) 3.378e-04  2.935e-05   11.51   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.04266 on 182 degrees of freedom
Multiple R-squared:  0.4688,    Adjusted R-squared:  0.463
F-statistic: 80.32 on 2 and 182 DF,  p-value: < 2.2e-16
```

Model 2:

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.028e+00  5.096e-01  13.791  < 2e-16 ***
strike1     -9.948e-02  7.460e-03 -13.336  < 2e-16 ***
expiry1     -4.452e-01  1.290e-01  -3.452 0.000693 ***
I(strike1^2) 3.583e-04  2.725e-05  13.151  < 2e-16 ***
I(expiry1^2) 7.193e-01  2.937e-01   2.449 0.015268 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03927 on 180 degrees of freedom
Multiple R-squared:  0.5548,    Adjusted R-squared:  0.5449
F-statistic: 56.08 on 4 and 180 DF,  p-value: < 2.2e-16
```

Model 3:

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     6.9125846  0.5166513  13.380   <2e-16 ***
strike1        -0.0972165  0.0076538 -12.702   <2e-16 ***
expiry1        -0.8787460  0.3623504  -2.425   0.0163 *
I(strike1^2)    0.0003479  0.0000284  12.251   <2e-16 ***
I(expiry1^2)    0.6730703  0.2953746   2.279   0.0239 *
strike1:expiry1 0.0033215  0.0025950   1.280   0.2022
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0392 on 179 degrees of freedom
Multiple R-squared:  0.5589,    Adjusted R-squared:  0.5465
F-statistic: 45.35 on 5 and 179 DF,  p-value: < 2.2e-16
```

Model 4 (with higher degree terms):

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)    -1.008e+02  4.737e+01  -2.129  0.03463 *
strike1         3.353e+00  1.421e+00   2.359  0.01939 *
expiry1        -8.706e-01  3.290e-01  -2.646  0.00887 **
I(strike1^2)   -4.072e-02  1.594e-02  -2.554  0.01149 *
I(strike1^3)    2.155e-04  7.924e-05   2.719  0.00719 **
I(strike1^4)   -4.208e-07  1.473e-07  -2.857  0.00478 **
strike1:expiry1 5.495e-03  2.403e-03   2.286  0.02340 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03599 on 178 degrees of freedom
Multiple R-squared:  0.6302,    Adjusted R-squared:  0.6178
F-statistic: 50.56 on 6 and 178 DF,  p-value: < 2.2e-16
```

ANOVA table

```
Analysis of Variance Table

Response: iv1
                Df   Sum Sq  Mean Sq  F value    Pr(>F)
strike1          1 0.051385 0.051385  33.4438 3.209e-08 ***
expiry1          1 0.022584 0.022584  14.6991 0.0001745 ***
I(strike1^2)     1 0.262684 0.262684 170.9680 < 2.2e-16 ***
I(expiry1^2)     1 0.009251 0.009251   6.0207 0.0150960 *
strike1:expiry1  1 0.002517 0.002517   1.6383 0.2022163
Residuals      179 0.275025 0.001536
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```