THE LOCAL VOLATILITY SURFACE

Abstract

In this paper we describe a method to smooth the implied volatility surface and demonstrate how to build local volatility surface from market data.

Haimao Yan

Haimao.yan77@qmail.cuny.edu

Contents

1 Introduction

While the implied volatility of an option is the market's estimate of the average future stock volatility during the life of that option, the local volatility is the market's estimate of stock volatility at a particular future time and price level. The implied volatilities can be identified by quoted option prices from market. Those implied volatilities for a range of strikes K and expirations T constitutes an implied volatility surface. We can extract these implied volatilities for future local volatility as a function of price level S and time t from the spectrum of available options prices as quoted in market. In other word, the implied volatilities surface implies an obscure, hitherto hidden, local volatility surface.

However, there is difficulty with extracting process. The available quoted options are not continually distributed on strikes K and expirations T. This causes the implied volatility surface directly observed from market is not smooth, and result in failed to calculate the local volatility.

In this paper we describe a method to smooth the implied volatility surface and demonstrate how to build local volatility surface from available stock options' prices on market. The process consists of calculating the implied volatility by Black-Scholes model, fitting implied volatility smile by SABR model, smooth the implied volatility surface by polynomials, and producing the local volatility surface by Dupire's formula. In appendix, we present several key technologies used in programming this process.

2 Model Description

2.1 Black-Scholes model

The Black-Scholes model assumes the stock price follow geometric Brownian motion with constant coefficients:

$$\frac{dS}{S} = \mu dt + \sigma dW$$

In the simplest case of a call, the PDE for the option price is:

$$\frac{\partial \mu}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 \mu}{\partial S^2} + \mathrm{rS}\frac{\partial \mu}{\partial S} - \mathrm{r\mu} = 0$$

with the final condition as:

$$\mu(S,T) = \max(S-K,0)$$

This problem can be solved analytically and the solution is:

$$\mu(S,T) = SN(d_1) - e^{-r(T-t)}KN(d_2) \qquad \textbf{(EQ 1)}$$

Where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}, \quad d_2 = d_1 - \sigma\sqrt{T-t}$$

This gives us a mapping: $\mu = \mu(S,t;K,T;\ \sigma,r)$ from the asset price S and the parameters to the option price, where μ is a quoted price, S is spot price, t is current date, T is the expiry date, K is the strike price, r is interest date, σ is stock volatility. All the parameters can be observed from market except the σ. The volatility can be identified from μ and is called implied volatility.

From the simplest call option, we defined the implied volatility to be the value of the σ, which substituted into the BS formula gives the price value of μ.

2.2 SABR model

The SABR (stochastic, α, β, and ρ) model is an extension of Black's model. It produces an estimate of the implied volatility curve, which is subsequently used as an input in Black's model to price swaps, caps, and other derivatives.

The SABR model is described by the following 3 equations:

$$df_t = \alpha_t f_t^{\beta} dW_t^1$$
$$d\alpha_t = v\alpha_t dW_t^2$$
$$E[dW_t^1 dW_t^2] = \rho dt$$

In these equations, f_t is the forward price, α_t is the volatility, and W_t^1 and W_t^2 are correlated Brownian motions, with correlation ρ. The model has 4 parameters: α, β, ρ, and v.

- ρ the correlation between the Brownian motions
- β determine the degree of the smile
- v parameter, volatility of volatility, determines the skew.
- α determines the at-the-money forward (ATM) volatility.

The case β = 0 produces the stochastic normal model, β = 1 produces the stochastic lognormal model, and β= ½ produces the stochastic CIR model.

The SABR model has a unique feature that allows us to compute the implied volatility directly for a given strike:

(EQ 2)

$$\sigma_B(F,K) = \left[\frac{\alpha}{(FK)^{(1-\beta)/2}\left(1+\frac{(1-\beta)^2}{24}\left(\ln\frac{F}{K}\right)^2+\frac{(1-\beta)^4}{1920}\left(\ln\frac{F}{K}\right)^4\right)}\frac{z}{\chi(z)}\right] * \left[1+\left(\frac{(1-\beta)^2}{24}\frac{\alpha^2}{(FK)^{1-\beta}}+\frac{1}{4}\frac{\alpha\beta\rho v}{(FK)^{(1-\beta)/2}}+\frac{2-3\rho^2}{24}v^2\right)\right]\mathrm{T}$$

where

$$z = \frac{v}{\alpha}(FK)^{(1-\beta)/2}\ln\frac{K}{F}$$

$$\chi(z) = \ln\left[\frac{\sqrt{1-2\rho z+z^2}+z-\rho}{1-\rho}\right]$$

Once the parameters α, β, ρ, and ν are estimated, the implied volatility σ_B is a function only of the forward price F and strike K.

2.3 Local Volatility Model

The Dupire formula enables us to deduce the volatility function in a local volatility model from quoted put and call options in the market. The formula as below:

(EQ 3)

$$\sigma_{loc}^2(K,T) = \frac{\frac{\partial C}{\partial T}}{\frac{1}{2}K^2\frac{\partial^2 C}{\partial K}}$$

Where $C = C(t, T, F, K)$ is time –t value of a European call option on forward price$F(T)$.

The Dupire's local volatility formula in term of implied volatility function is given by:

$$\sigma_{loc}^2(K,T) = \frac{\sigma_{imp}^2 + 2\sigma_{imp}(T-t)(\frac{\partial\sigma_{imp}}{\partial T} + (r-d)K\frac{\partial\sigma_{imp}}{\partial K})}{\left(1 + Kd_1\frac{\partial\sigma_{imp}}{\partial K}\sqrt{T-t}\right)^2 + \sigma_{imp}K^2(T-t)\left[\left(\frac{\partial^2\sigma_{imp}}{\partial K^2}\right) - d_1\left(\frac{\partial\sigma_{imp}}{\partial K}\right)^2\sqrt{T-t}\right]} \quad \text{(EQ 4)}$$

Where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma_{imp}^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

and implied volatility function σ_{imp}=$\sigma_{imp}(T,K)$ is assumed to be sufficiently smooth.

3 Producing Local Volatility Surface

3.1 Implied Volatilities

Through Black-Scholes model (EQ 1), the implied volatility can be identified from option prices, which can be observed from market. According to the assumptions of the Black-Scholes model, the implied volatility should be the same from all strikes *K* and times *T*. However, in market, the volatility depends on both. Thus, we need to calculate all the options' volatilities $\sigma_{BS}(K,T)$, which is volatility surface.

When calculate the implied volatility surface, few things we need to highlight:

1. For option's market price, we should always use the average of option's bid and ask price.
2. Theoretically, the implied volatility should be same for Put and Call on same strike and time, but at market, because of trading liquidity, the volatility is not same. Since the OTM option is more active, we should always use out-of-money option's volatility.

3. For the out-the-money put/call, given the maturity, the option price would decrease along with decreasing / increasing of strike. If the option price is close to zero, we need to stop calculate the volatility of option in the rest of strikes.

Because the strikes are various in different maturities and the maturities are not calibration, the implied volatilities we get are not smooth.

3.2 Fitting Implied Volatilities

Once we get the market implied volatilities, we choose to use the SABR model (EQ 2) to fit the volatility smile of each maturity. At this stage, we are given a set of maturities $(T_i)_{i=1..i}$ and strikes $\left(K_j\right)_{j=1..j}$ along with the corresponding market volatility $M = \left(\sigma_{BS}^M\left(T_i, K_j\right)\right)_{i,j}$ where σ_{BS}^M is Black-Scholes implied volatility with strike K_j and maturityT_i.

The calibration begins with choosing β either by empirical analysis of the asset price and the σ_{ATM} or by setting $\beta = 0$ for a normal process or $\beta = 1$ for a lognormal process. In this paper, we use the $\beta = 1/2$, which is stochastic CIR model.

Next, with β chosen, we have two choices in calibrating the other 3 parameters:

1. Estimate α, ρ, and v directly by minimizing the errors between the volatilities calculated by model and market volatilities σ_{BS}^M with identical maturity T. Hence, we can use SSE (Sum of Square Errors), which produces:

$$(\hat{\alpha}, \hat{\rho}, \hat{v}) = arg \min_{\alpha,\rho,v} \sum_i \left\{\sigma_i^{mkt} - \sigma_B(F_i, K_i; \alpha, \rho, v)\right\}^2$$

   We then use α, β, ρ, and v in equation (3) to obtain σ_B

2. Estimate ρ, and v directly, and infer α from ρ, and v, and the at-the-money volatility σ_{ATM}

Base on the research of F. Rouah, both methods produce a set of implied volatilities that fit the market volatilities reasonably well. As the second method would take a longer time to estimate, we would choose the first method to estimate parameters directly.

Another important thing need to mention: we found that using the Sum of Square of Error Percent to estimate the parameters instead of SSE would have better fitting, in the case that bad data existed in real market. The Sum of Square of Error Percent formula as below:

$$(\hat{\alpha}, \hat{\rho}, \hat{v}) = arg \min_{\alpha,\rho,v} \sum_i \left\{ \frac{\sigma_i^{mkt} - \sigma_B(F_i, K_i; \alpha, \rho, v)}{\sigma_i^{mkt}} \right\}^2 \tag{5}$$

Then, we fit SABR model per each maturity separately. By calibration, we impose the following restriction:

a) v : As the SABR model is based on assumption of small time scale of volatility of volatility parameters $v^2T < 1$, we impose the $v \sim \left(0,\ \sqrt{1/T}\right)$
b) α : since α determines the at-the-money forward (ATM) volatility, we impose $\alpha > 0$
c) ρ : As this is correlation, we set the $\rho \sim (-1,\ 1)$

Once we estimated the parameters, we can calculate the fitted implied volatilities. The multiple volatility smiles on different maturities are plotted as below:

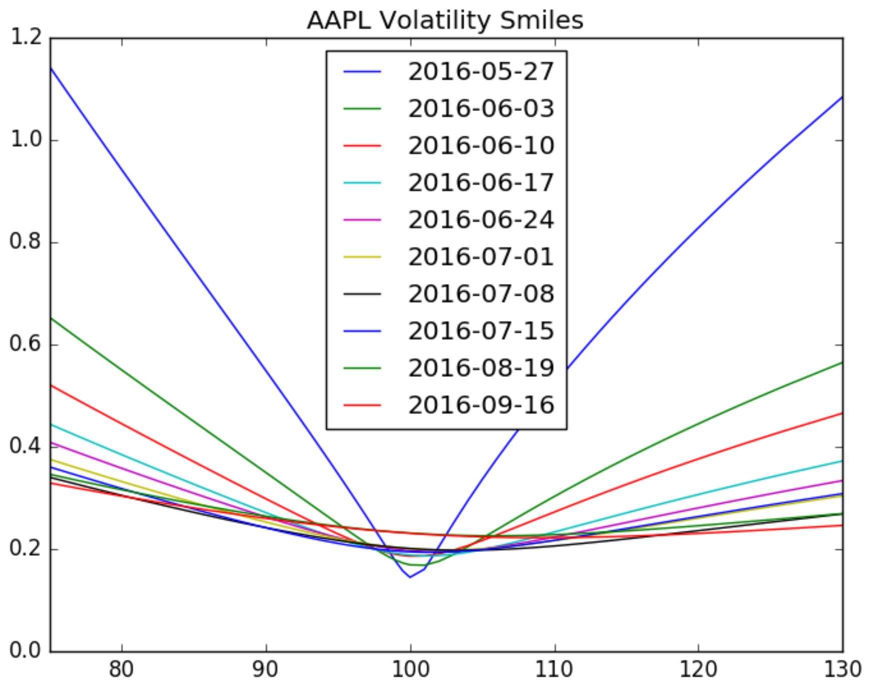


Figure 1: the implied volatility smile for AAPL

After the calibration, we obtain a set of model implied parameters $\Pi^* = \left(\left(\hat{\alpha}(T_i), \hat{\rho}(T_i), \hat{\upsilon}(T_i)\right)\right)_i$ representing a parametrization of market volatilities of each maturity.

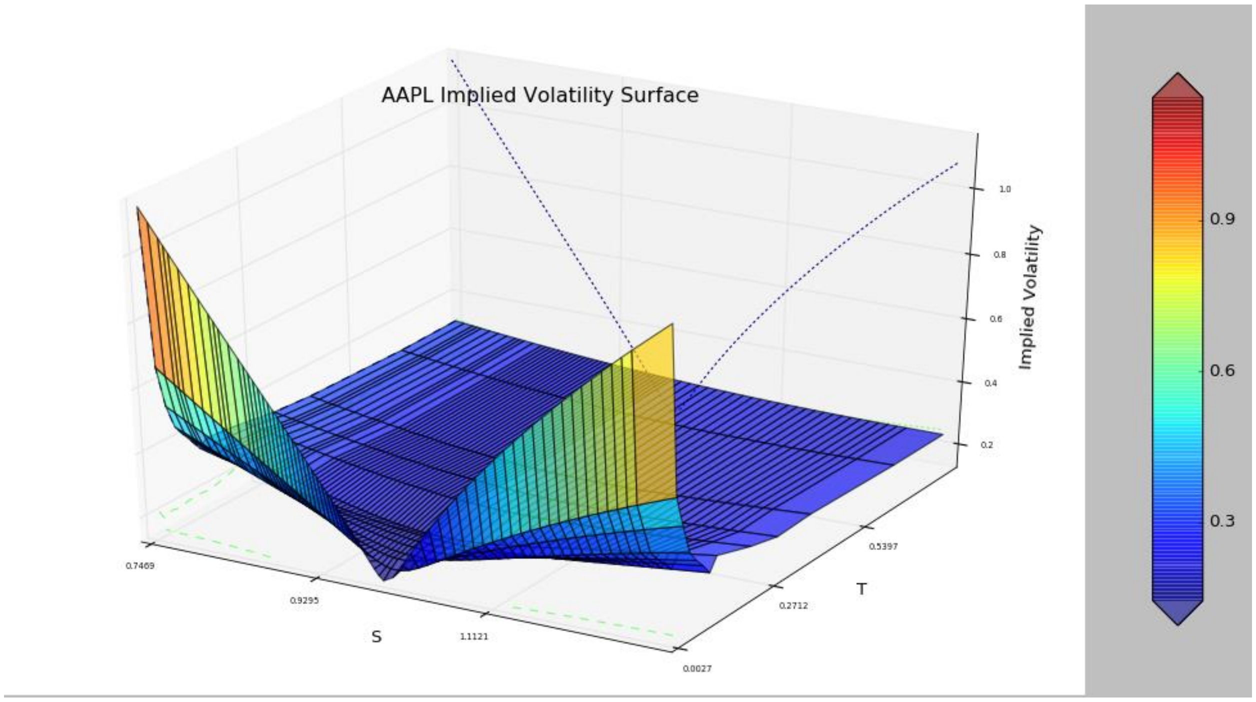


Figure 2: the unsmoothed market implied volatility surface for AAPL

3.3 Smooth Implied Volatility Surface

To produce a smooth local volatility, we need to smooth implied volatility surface sufficiently. So, once separate maturities are fitted, the problem becomes how to stitch them together in order to produce a smooth volatility surface. To deal with this issue appropriately, we propose the parameter interpolation by polynomials in time direction.

Base on research of Artur Sepp (2007), the dimensions of these variables and parameters are as follows:

$$[\alpha] = T^{-\frac{1}{2}}, \qquad [\rho] = T^{-1}, \qquad [\nu] = T^{-\frac{1}{2}}$$

We use the following method of polynomial interpolation:

$$\upsilon^{-2}(T) = a_0 + a_1T + \cdots + a_kT^k$$

$$\rho^{-1}(T) = b_0 + b_1T + \cdots + b_hT^h$$

$$\upsilon^{-2}(T) = c_0 + c_1T + \cdots + c_lT^l$$

Then, we can use the available empirical data points (see Figure 3) in $\Pi^* = \left(\left(\widehat{\alpha}\,(T_i), \widehat{\rho}(T_i), \widehat{\upsilon}(T_i)\right)\right)_i$ to calculate coefficients$(a_k)_{k=0,1\ldots k}$, $(b_h)_{h=0,1,\ldots h}$ and $(c_l)_{l=0,1,\ldots l}$ under certain degree of freedom.

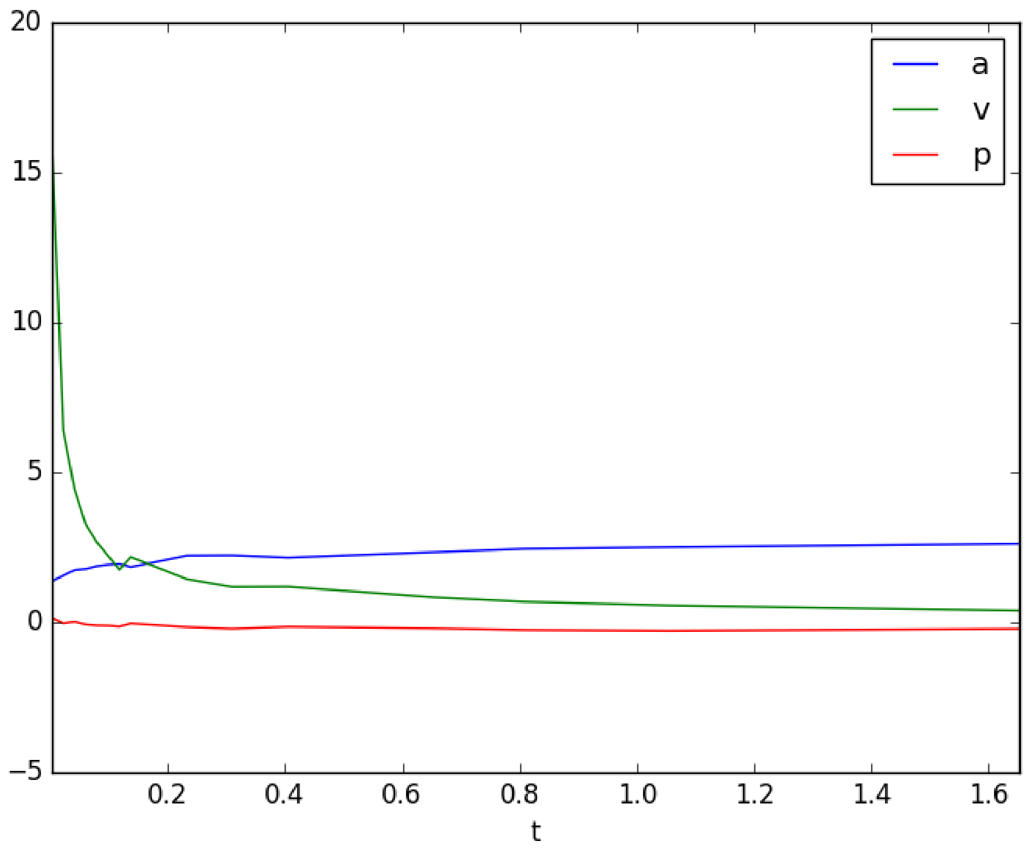


Figure 3: the estimated parameters for AAPL

There is one thing need to special attention: the choice of degree for interpolating polynomials, $k, h, and\ l$, is of empirical nature. Higher degree results in better fit to Π^* and ultimately to market data M, but it might produce oscillation if data in Π^* are not smooth. In the case of implied volatilities under certain maturity are abnormal (like extremely low), which would cause α and ρ severely shock, we need to drop those maturity's data points and recalculate the polynomials. Based on our test, the degree of interpolating $k, h, and\ l$ usually is between $\left[\frac{C}{3}, \frac{C}{2}\right]$, c is count of unique maturities.

Finally, given the interpolated values of $\alpha(T)$, $\rho(T)$, and $v(T)$, we calculate the volatilities for the other strikes and times, thus we obtain the smooth implied volatility surface $M = \left(\sigma_{BS}^{M}(T_i, K_j)\right)_{i,j}$.

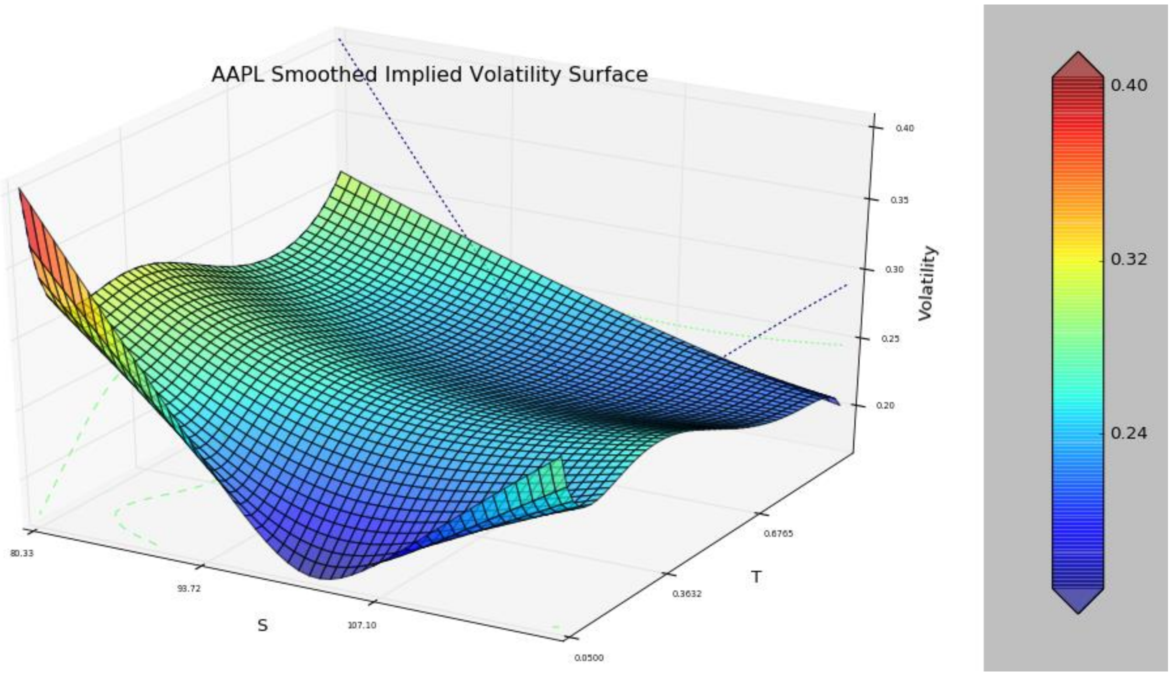


Figure 4: the smoothed market implied volatility surface for AAPL

3.4 Compute Local Volatility Surface

Once we get the smooth implied volatility surface, we can transfer the formula (EQ 4) to continuous form of Dupire's formula to calculate the local volatility surface.

$$\sigma_{loc}^2(K,T) = \frac{\sigma_{imp}^2 + 2\sigma_{imp}(T-t)(\frac{\Delta\sigma_{imp}}{\Delta T} + (r-d)K\frac{\Delta\sigma_{imp}}{\Delta K})}{\left(1 + Kd_1\frac{\Delta\sigma_{imp}}{\Delta K}\sqrt{T-t}\right)^2 + \sigma_{imp}K^2(T-t)\left[\left(\frac{\Delta^2\sigma_{imp}}{\Delta K^2}\right) - d_1\left(\frac{\Delta\sigma_{imp}}{\Delta K}\right)^2\sqrt{T-t}\right]} \quad \text{(EQ 5)}$$

In order to calculate sigma's derivative with respect to the strike price K and the time T in the equation above, we use the finite difference approximation.

The first order derivative:

$$\frac{\Delta\sigma_{imp}}{\Delta T} = \frac{\sigma_{imp}(T_i, k) - \sigma_{imp}(T_{i-1}, k)}{T_i - T_{i-1}} \qquad \text{(EQ 6)}$$

$$\frac{\Delta\sigma_{imp}}{\Delta K} = \frac{\sigma_{imp}(T_i, k_j) - \sigma_{imp}(T_i, k_{j-1})}{K_j - K_{j-1}}$$

The second order derivative:

$$\frac{\Delta^2\sigma_{imp}}{\Delta K^2} = \frac{\sigma_{imp}(T_i, k_j) - 2\sigma_{imp}(T_i, k_{j-1}) + \sigma_{imp}(T_i, k_{j-2})}{(K_j - K_{j-2})^2} \qquad \text{(EQ 7)}$$

Finally, based on smooth implied volatility, we calculate the local volatility surface under price K and time T dimension, $M = \left(\sigma_{loc}^{M}(T_{i-1}, K_{j-2})\right)_{i,j}$.

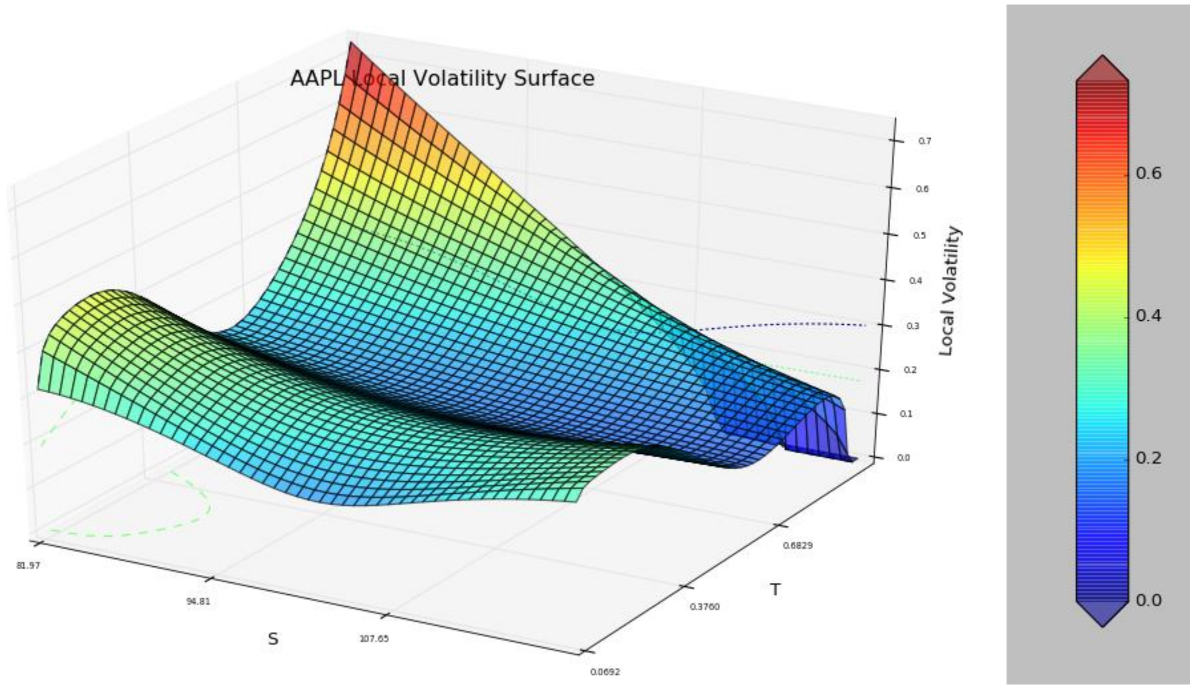


Figure 5: the Local volatility surface for AAPL

4 Conclusion

We have developed a process to build a smooth local volatility surface from quoted option data, which includes fitting a smooth implied volatility surface to market data and deriving a local volatility surface from the surface. Based on our test for stock AAPL and equity index SPY, the model works reasonable well. However, there are still some issues, like the smooth volatility surface should be the arbitrage free, how to verify the local volatility surface etc., need to further discuss.

Appendix: Programming Technology

In this appendix, we represent the programming technology that used to develop the whole process. The source code can be accessed at https://github.com/soyatech/localvolatility.

1. **Programming language**

   The application is developed in Python 3.5 (https://www.python.org/). The develop environment is Jupyter Notebook (http://jupyter.org/).

2. **Installed Package**

   a. ***Pandas***: is an open source library providing high-performance, easy-to-use data structures and data analysis tools for Python programming language. (Download http://pandas.pydata.org/.)

   b. ***Scipy***: is an open-source library providing many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. (Download: https://www.scipy.org/scipylib/index.html.)

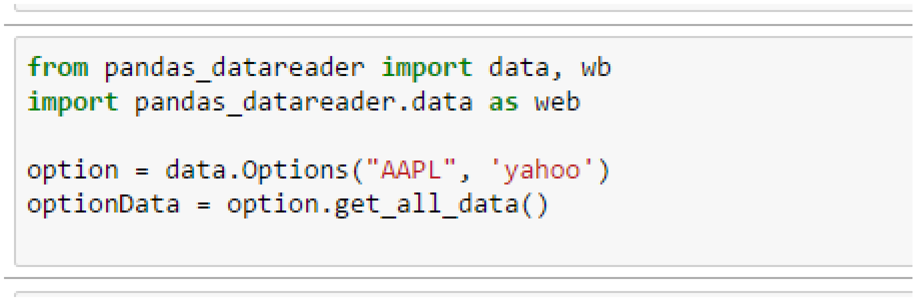
   c. ***Matplotlib***: is a python plotting library providing plots, histograms, power specutra, bar charts, errorcharts, scatterplots, etc. (Download: http://matplotlib.org/.)

   d. ***Numpy***: is a fundamental package for scientific computing with Python. It contains a powerful N-dimensional array object, sophisticated functions, useful linear algebra, Fourier transform, and random number capabilities. (Download: http://www.numpy.org/)

3. **The Key Technologies**

   a. Download option data from Yahoo

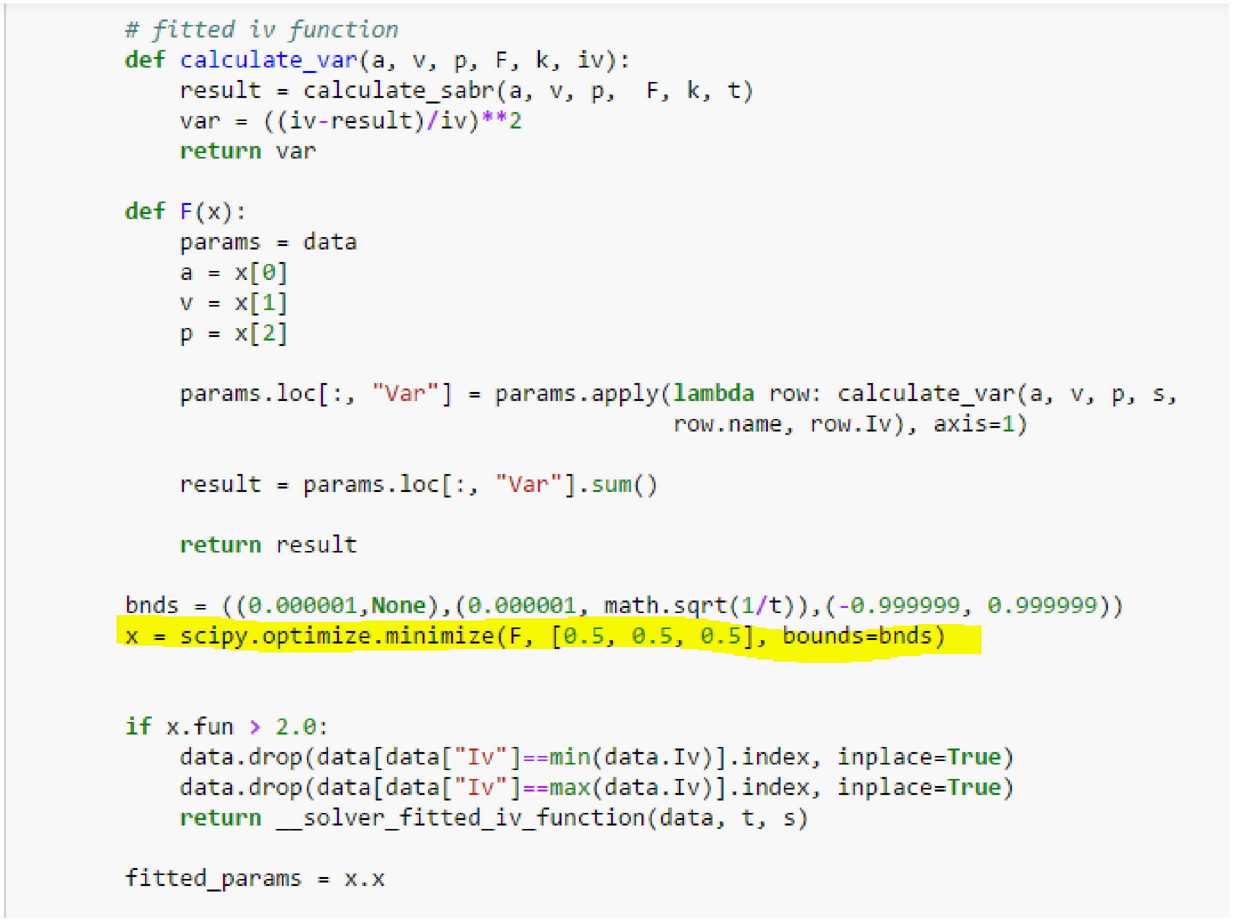
   The pandas provide a remote access function, which can easily download data from Yahoo, Google, etc. We use this function to download the function data from Yahoo finance. The sample code as below:



```
from pandas_datareader import data, wb
import pandas_datareader.data as web

option = data.Options("AAPL", 'yahoo')
optionData = option.get_all_data()
```

b. ***Solve the parameters by SSE***

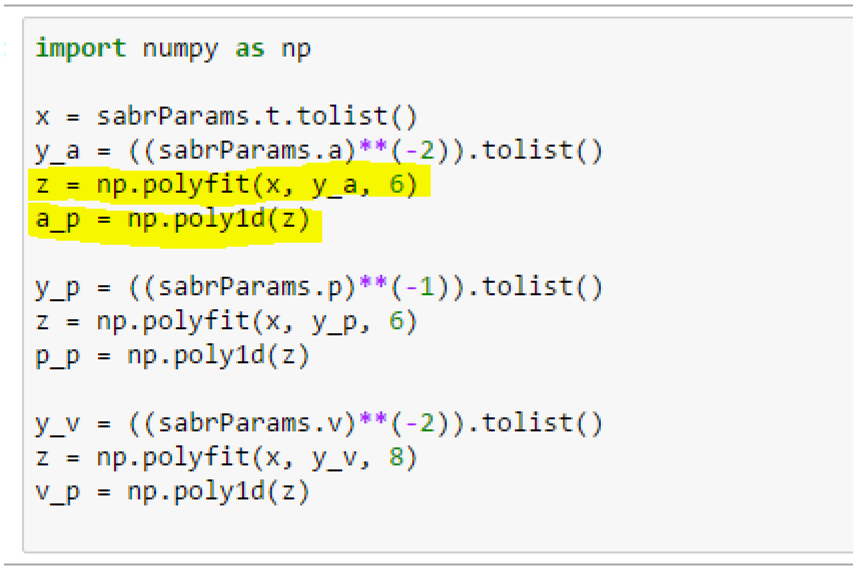
Given the market data, we use Scipy.optimize function to solve the SABR parameters by least sum of square error. The sample code as below:



```
# fitted iv function
def calculate_var(a, v, p, F, k, iv):
    result = calculate_sabr(a, v, p,  F, k, t)
    var = ((iv-result)/iv)**2
    return var

def F(x):
    params = data
    a = x[0]
    v = x[1]
    p = x[2]

    params.loc[:, "Var"] = params.apply(lambda row: calculate_var(a, v, p, s,
                                        row.name, row.Iv), axis=1)

    result = params.loc[:, "Var"].sum()

    return result

bnds = ((0.000001,None),(0.000001, math.sqrt(1/t)),(-0.999999, 0.999999))
x = scipy.optimize.minimize(F, [0.5, 0.5, 0.5], bounds=bnds)


if x.fun > 2.0:
    data.drop(data[data["Iv"]==min(data.Iv)].index, inplace=True)
    data.drop(data[data["Iv"]==max(data.Iv)].index, inplace=True)
    return __solver_fitted_iv_function(data, t, s)

fitted_params = x.x
```

c. ***Solve the Polynomial coefficients***

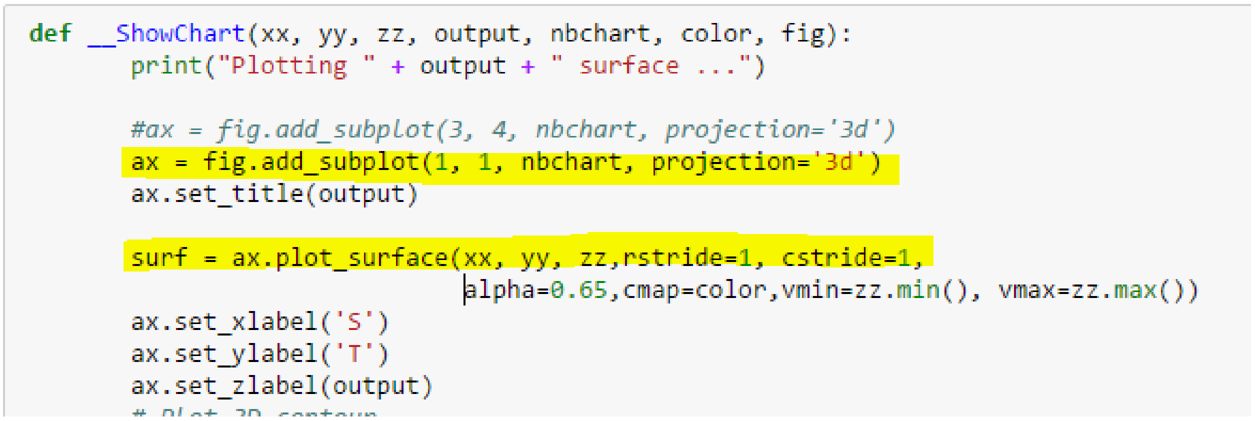
We use the Numpy Polyfit function to solve the Polynomial coefficents. The numpy.polyfit can fit a polynomial `p(x) = p[0] * x**deg + ... + p[deg]` of degree deg to points (x, y). The function will return a vector of coefficients that minimize the squared error. The sample code as below:



```
import numpy as np

x = sabrParams.t.tolist()
y_a = ((sabrParams.a)**(-2)).tolist()
z = np.polyfit(x, y_a, 6)
a_p = np.poly1d(z)

y_p = ((sabrParams.p)**(-1)).tolist()
z = np.polyfit(x, y_p, 6)
p_p = np.poly1d(z)

y_v = ((sabrParams.v)**(-2)).tolist()
z = np.polyfit(x, y_v, 8)
v_p = np.poly1d(z)
```

d. Plot the volatility surface and line chat

We use Matplotlib to plot the volatility surface and volatility smiles. The plot surface code as below:



```
def __ShowChart(xx, yy, zz, output, nbchart, color, fig):
        print("Plotting " + output + " surface ...")

        #ax = fig.add_subplot(3, 4, nbchart, projection='3d')
        ax = fig.add_subplot(1, 1, nbchart, projection='3d')
        ax.set_title(output)

        surf = ax.plot_surface(xx, yy, zz,rstride=1, cstride=1,
                                    alpha=0.65,cmap=color,vmin=zz.min(), vmax=zz.max())
        ax.set_xlabel('S')
        ax.set_ylabel('T')
        ax.set_zlabel(output)
        # Plot 2D contour
```

References

[1] Dupire B, (1994), "Pricing with a smile," *Risk January*, 1820.

[2] Emanuel D, Iraj K, Joseph Zou, (1995), "The Local Volatility Surface" *Goldman Sachs Quantitative Strategies Research Notes*

[3] Fabrice Douglas Rouah, (2007), "The SABR Model", *www.FRouah.com*

[4] Artur Sepp, (2007), "Using SABR model to produce smooth local volatility surfaces", *http://kodu.ut.ee/~spartak/paper*

[5] Patrick S. Hagan, (2002), "Managing Smile Risk", *WILMOTT* magazine, 84-102