CY-TECH

Model calibration and simulation

ING3 IFI

Dupire's equation and calibration of local volatility models

Author:
Farès Djerourou

January 2023

Contents

1 Introduction

The Black-Scholes equation gives us the price of the option as a function of the value of the underlying asset S at time t for a set time to maturity T and strike price K.
Evidence suggests that the volatility of the underlying asset depends on T and K, this phenomena produces the well-known volatility smile associated with implied volatility.
The Dupire equation on the other hand takes a different approach to the problem, it answers the question "For a set asset price S at time t, how would the value of the option change if we change the values of T and K?".

In this paper, we will numerically solve the Dupire equation using a Crank-Nicolson finite difference scheme, we will assume the volatility component is local and depends on K, we will then use an optimisation algorithm to find the local volatility function's parameters that give us the market prices of the option. We will consider the CEV and Gatheral models for local volatility.

We will follow by using a cubic spline algorithm to interpolate the volatility smile of the S&P call data, we will then use the interpolated graphic to deduce the local volatility with the help of relationship between local and implied volatilities which is given to us by the BS and Dupire equations.

2 Solving the Dupire Equation

2.1 The Dupire Equation

The Dupire equation with respect to the variables T and K is given by the following formula:

$$\begin{cases} \dfrac{\partial V}{\partial T} + rK\dfrac{\partial V}{\partial K} - \dfrac{1}{2}\sigma^2_{local}(K,T)K^2\dfrac{\partial^2 V}{K^2} = 0 \\ \quad V(T=0,K;t_0,S_0) = max(S_0-K,0) \end{cases} \tag{1}$$

Where $\sigma_{local}(K,T)$ is the local volatility.

We will now discretize T and K to numerically solve the equation using the implicit Crank-Nicolson scheme:

$$B_i V_{i-1}^{n+1} + D_i V_i^{n+1} + A_i V_{i+1}^{n+1} = C_i^n \tag{2}$$

2.2 Numerical Application

To solve the implicit scheme and find the solution to the Dupire equation, we will need to use Thomas' algorithm at each step iteration to solve the corresponding linear system. We will first consider the case were $\sigma_{local}(K, T)$ follows a CEV model.

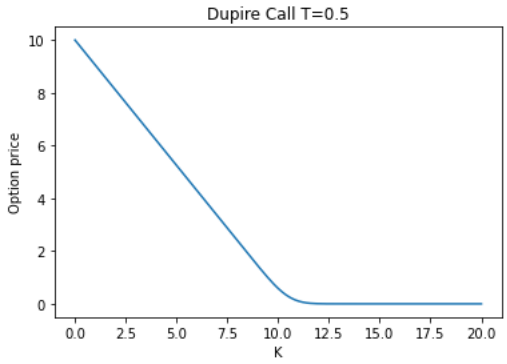

Figure 1: Dupire Call $T = T_{max}$

This is the option's price when $T = T_{max}$, we see that the option's value decreases as K increases, the breaking point being slightly above $K = 10$ where the option loses its value, the options still holds some value at $K = 10$ because there is still a possibility for it to finish in-the-money.

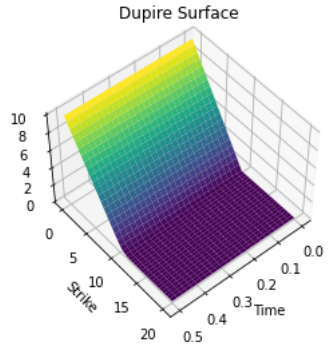

Figure 2: Call Option's Surface

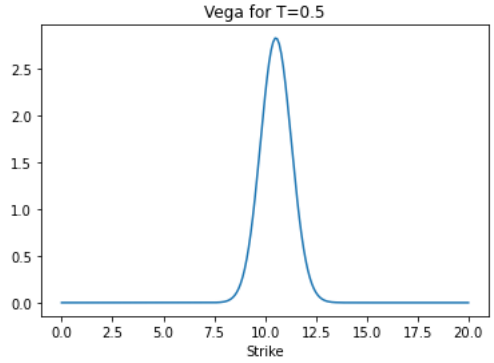

Figure 3: Vega Call $T = T_{max}$

We will no take a look at the Vega of the option.

The option's Vega was calculated using a finite difference approximation, the step was taken as $h = 0.01$.

Vega gives the option's price sensitivity to volatility, we can see that the option's price is the most sensitive to volatility when the $K = 10$, which makes sense since $K = 10$ is when the option is at-the-money and any movement in the underlying means it will either be in-the-money or at-the-money.

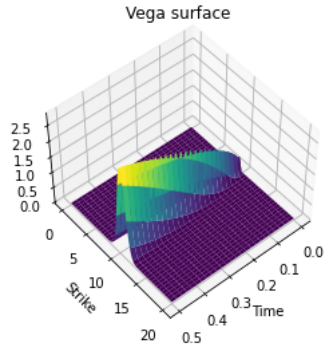

Figure 4: Surface of the option's Vega

3 Local Volatility Calibration

3.1 Levenberg-Marquardt

In this section we will calibrate the constant elasticity of variance model as well as the Gatheral local volatility model using the Levenberg-Marquardt algorithm and the market prices of the option for some K values. To calibrate the model is to find the parameters minimizing our objective function, there are a lot of optimisation algorithms someone could use to achieve that.

Gradient Descent

$$x_{n+1} = x_n - \lambda J_r^T r(x_n)$$

Gauss-Newton

$$x_{n+1} = x_n - (J_r^T J_r)^{-1} J_r^T r(x_n)$$

The advantage of the Gauss-Newton algorithm over Gradient Descent lies in the fact that $J_r^T J_r$ approximates the Hessian of the function $H \approx J_r^T J_r$ instead of relying on the λ constant at each step, and thus gives faster convergence.
There's a problem with Gauss-Newton's algorithm nevertheless, the Hessian matrix can become ill-conditioned, that is the case, for example, when the initial guess if far from the target solution of the problem, here enters Levenberg-Marquardt.

Levenberg Marquardt

$$x_{n+1} = x_n - (J_r^T J_r + \lambda D)^{-1} J_r^T r(x_n)$$

Unlike the previous algorithms, with the adjustment λD to the Hessian approximation, Levenberg-Marquardt can (often) still find the solution even if the initial guess is far from it, it is more robust, although it might be slower than Gauss-Newton for well-behaved functions.

3.2 Calibration in the CEV model

The objective function that we are looking to minimize is:

$$\sum_{p=1}^{6} |V^{market}(T_p, K_p) - V^{Dupire}(T_p, K_p, \sigma_{local}(\beta_1, \beta_2)|^2$$

We recall that the local volatility in the CEV model is given by

$$\sigma_{local}(\beta_1, \beta_2) = \frac{\beta_1}{K^{\beta_2}}$$

We will be using Levenberg-Marquardt as our our optimisation algorithm. The starting values for the parameters will be $\beta_1 = \beta_2 = 1$.

After a Jacobian calculation and a quick Python implementation we find the following values after 28 iterations!

$$\beta_1 = 1.69949217$$
$$\beta_2 = 0.79986239$$

3.3 Calibration in the Gatheral model

The objective function that we want to minmize is similar to the previous case

$$\sum_{p=1}^{6} |V^{market}(T_p, K_p) - V^{Dupire}(T_p, K_p, \sigma_{local}(\beta_1, \beta_2)|^2$$

The local volatility in the Gatheral model is given by

$$\sigma_{local}(S, t) = b(\rho(S - m) + \sqrt{(S - m)^2 + a^2}$$

Here, we will also be using Levenberg-Marquardt as our our optimisation algorithm. But first we will assume that b and ρ are known, in fact we will take

$$b = 0.05$$
$$\rho = 0.1$$

And we will be looking for $a = \beta_1$ and $b = \beta_2$ The starting values for the parameters will also be set to $\beta_1 = \beta_2 = 1$.

After a (another!) Jacobian calculation and a thankfully similar quick Python implementation we find the following values for our parameters after 35 iterations!

$$\beta_1 = 10.20270711$$
$$\beta_2 = 12.00874008$$

Let's look at that calibrated local volatility now shall we?

The Vega surface in this local volatility model seems more spread out, meaning the option's price is relatively more sensitive to the volatility for a greater range of strike prices.

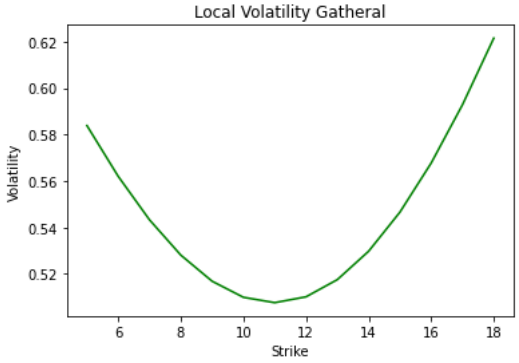

Figure 5: $K \to b(\rho(K-m) + \sqrt{(K-m)^2 + a^2}$

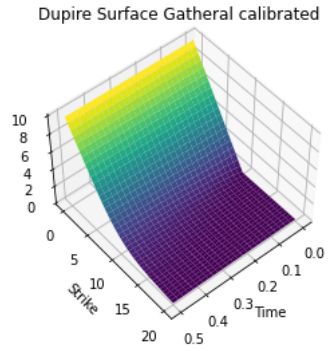

Figure 6: Calibrated Dupire surface Gatheral

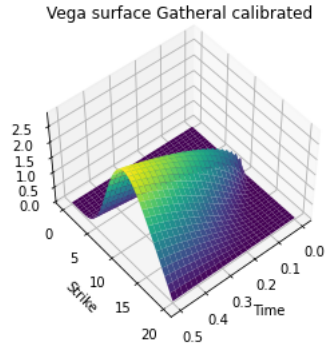

Figure 7: Calibrated Vega surface Gatheral

4 Cubic Spline Interpolation

4.1 Call-Put Parity

Our aim in this section is to use the call-put parity to find the initial value of the underlying asset as well as the dividend rate

$$S_0 e^{-qT} = Call(S_0, T, K) - Put(S_0, T, K) + Ke^{-rT}$$

We will be using the S&P Index call data provided in the accompanying document. From the call-put parity we deduce

$$1 \leq \mathrm{j} \leq 8$$

$$ln(\hat{S}_j) = ln(S_0) - qT_j$$

With $\hat{S}_j$ representing the mean of the $S_j e^{-qT_j}$ values. We then perform a linear regression to find the values of S_0 and q based on our S&P data.

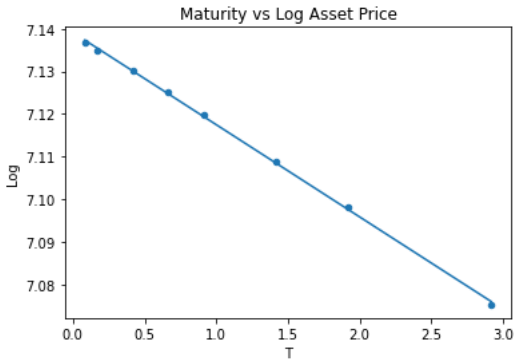

Figure 8: Linear regression

$$S0 = 1260.3666787091645$$
$$q = 0.02166466966128411$$

4.2 Cubic Spline for implied and local volatilities

Our goal here is to interpolate the implied volatility graphic using cubic spline functions. To do that we need to solve some equations!

Lets us consider $y_i'' = f''(x_i)$ as $f(x_i)$ is our cubic spline function. To find those values, we have to solve the following equations

$$\frac{(x_i - x_{i-1})}{(x_{i+1} - x_{i-1})} y_{i-1}'' + 2y_i'' + \frac{(x_{i+1} - x_i)}{(x_{i+1} - x_{i-1})} y_{i+1}'' = 6f[x_{i-1}, x_i, x_{i+1}] \tag{3}$$

$$f[x_{i-1}, x_i, x_{i+1}] = \frac{(y_{i+1} - y_i)(x_i - x_{i-1}) - (y_i - y_{i-1})(x_{i+1} - x_i)}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)(x_i - x_{i-1})}$$

In fact, finding $y_i'' = f''(x_i)$ is equivalent to finding the respective third degree polynomials because

$$P_i(x) = \frac{(x - x_i)^3}{(x_{i-1} - x_i)} \frac{y_{i-1}''}{6} + \frac{(x - x_{i-1})^3}{(x_i - x_{i-1})} \frac{y_i''}{6} +$$

$$(y_{i-1} - \frac{y_{i-1}''}{6}(x_i - x_{i-1})^2) \frac{(x - x_i)}{(x_{i-1} - x_i)} + (y_i - \frac{y_i''}{6}(x_i - x_{i-1})^2) \frac{(x - x_{i-1})}{(x_i - x_{i-1})}$$

With the help of the following snippet of code, we will solve (3) and get the y_i'' values we need.

```
def ThomasAlgorithm(N,A,B,C,D):
    X=np.zeros(N)
    for i in range(1,N):
        w=A[i]/B[i-1]
        B[i]=B[i]-w*C[i-1]
        D[i]=D[i]-w*D[i-1]
    X[N-1]=D[N-1]/B[N-1]
    for i in range(N-2,-1,-1):
        X[i]=(D[i]-C[i]*X[i+1])/B[i]

    return X
```

We will also use the relationship between local and implied volatilities to deduce the corresponding local volatility graphic.

$$\sigma_{local}(K,T) = \sqrt{\frac{2\frac{\partial V^{Dupire}}{T} + rK\frac{\partial V^{Dupire}}{K}}{K^2 \frac{\partial^2 V^{Dupire}}{K^2}}}$$

As we can see from the following graphics, the local volatility seems to be pretty sensitive to the interpolation, which might suggest that additional data and or assumptions on the volatility smile curve might be needed in order to get more reasonable results.

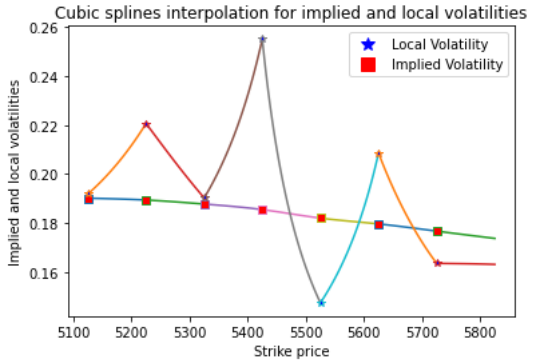

Figure 9: Implied And Local Vol Interpolation

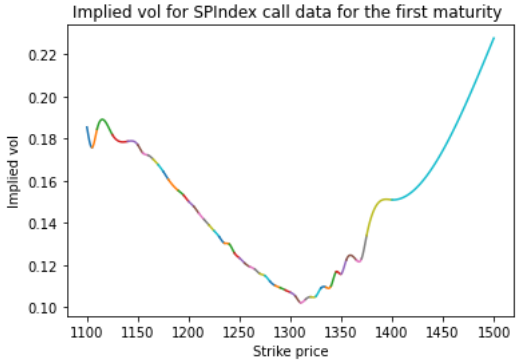

Figure 10: Implied Vol S&P

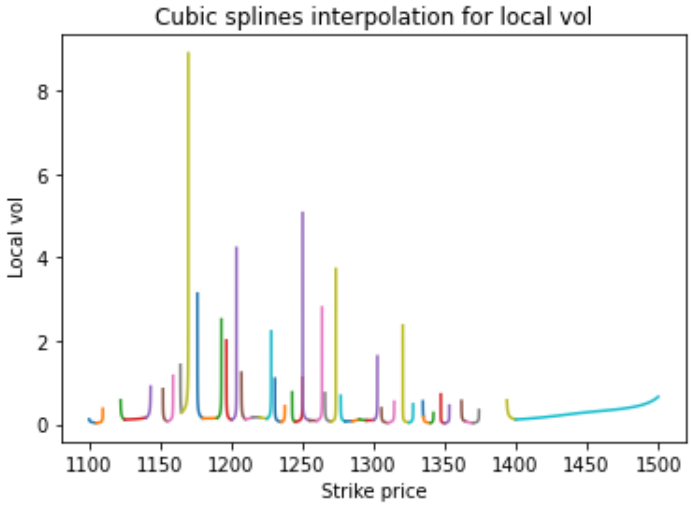

Figure 11: Local Vol Interpolation S&P