[Nom de la société]

Applied Data Science: Volatility smile

MS Finance and Big data 2020

Chris Emmanuel Ble Tanouhe
14/05/2020

TABLE DES MATIERES

PROJECT PHASE 1: IMPLIED VOLATILITY FROM OPTION'S PRICE

PROBLEMATIZATION

Implied volatility σ_{imp} is the volatility value σ that makes the Black-Scholes value of the option equal to the traded price of the option.

In the Black-Scholes model, the volatility parameter σ is the only parameter that cannot be directly observed. All other parameters can be determined through market data (Strike, forward, maturity). This being the case, the volatility parameter is the result of a numerical optimization technique given the Black-Scholes model.

The idea of this part is to derive the black and Scholes's volatility from the option prices provided in the excel sheet "Imply Volatilities from Prices USDJPY Smile.xlsx".

METHODOLOGY

Definition:

Black-Scholes model:

Consider C and P is the quoted value of a call and a put, respectively. We then define the value of a Black-Scholes option for a given maturity-strike pair as:

$$\begin{aligned} C(T,K) &= D(0,T)\left(F(0,T)N(d_1) - KN(d_2)\right) \\ P(T,K) &= D(0,T)\left(KN(-d_2) - F(0,T)N(-d_1)\right) \\ D(0,T) &= e^{-rT} \\ F(0,T) &= S(0)e^{rT} \end{aligned}$$

With D(0, T) : Discount factor ; F(0,T): Forward price; S(0): Spot price; T: maturity and expiry.

Greek Vega:

Vega is the first derivative of σ volatility and thus is an integral piece in the formulation of implied volatility. What follows is a quick derivation of Vega.

As Vega is the first derivative of volatility, its partial derivative takes the form $\frac{\partial C}{\partial \sigma}$. Therefore, we take the partial derivative of the Black-Scholes formula with respect to σ

$$vega(C) = \frac{\partial C}{\partial \sigma} = Se^{-q(T-t)}N'(d_1)\frac{\partial d_1}{\partial \sigma} - Ke^{-r(T-t)}N'(d_2)\frac{\partial d_2}{\partial \sigma}$$

We can then rearrange the formula into:

$$vega(C) = Se^{-q(T-t)}N'(d_1)\left(\frac{\partial d_1}{\partial \sigma} - \frac{\partial d_2}{\partial \sigma}\right)$$

We know $d_2 = d_1 - \sigma\sqrt{T-t}$, rearrange to get $d_1 - d_2 = \sigma\sqrt{T-t}$. Therefore,

$$\frac{\partial d_1}{\partial \sigma} - \frac{\partial d_2}{\partial \sigma} = \sqrt{T-t}$$

We can then replace $N'(d_1)$ as $N'(d_1) = \frac{1}{\sqrt{2\pi}}e^{\frac{-d_1^2}{2}}$ to get :

$$vega(C) = \frac{1}{\sqrt{2\pi}}Se^{-q(T-t)}Se^{\frac{-d_1^2}{2}}\sqrt{T-t}$$

For a non-dividend paying asset, the vega derives to the following:

$$S.N(d_1)\sqrt{T-t}$$

Newton algorithm:

Newton's method is a root finding method that uses linear approximation. We guess a solution x0 of the equation f(x)=0, compute the linear approximation of f(x) at x0 and then find the x-intercept of the linear approximation.

Let f(x) be a differentiable function. If x0 is near a solution of f(x)=0 then we can approximate f(x) by the tangent line at x0 and compute the x-intercept of the tangent line. The equation of the tangent line at x0 is:

$$y = f'(x_0)(x - x_0) + f(x_0)$$

The x-intercept is the solution x1 of the equation:

$$0 = f'(x_0)(x_1 - x_0) + f(x_0)$$

and we solve for x1:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

which (potentially) converges to a solution of the equation f(x)=0.

Implementation:

The method is basically to find for each given expiry the implied volatility using the newton's algorithm. So, we create a function on python called implied_vol which takes 7 inputs:
F: option's forward price
K: option's strike
T: time to expiry
V0: option's value
D: discount factor
sigma_init: implied volatility initial guess. We fixed it at 0.025
option_type: "Call" for a call option, "Put" for a put option and "Call or Put" for an option with a in the moneyness (ITM) equal 1

Newton approximation function

Entrée [462]:

```
def implied_vol(F,K,T,V0,D,sigma_init,option_type):

    #F: Foward price
    #K: strike price
    #T: time to maturity
    #D : Discount factor
    #sigma: volatility of underlying asset
    #V0 : Option's initial value

    S = F/D

    #Black and scholes option pricing:

    d1 = (np.log(F/K) + 0.5* sigma_init**2*T)/(sigma_init*np.sqrt(T))
    d2 = d1- sigma_init*np.sqrt(T)

    if option_type =='Call'or'Call or Put': fx = D*(F*si.norm.cdf(d1,0.0,1.0) - K*si.norm.cdf(d2,0.0, 1.0)) - V0
    if option_type =='Put': fx = D*(K*si.norm.cdf(-d2,0.0,1.0) - F * si.norm.cdf(-d1,0.0, 1.0)) - V0

    #Vega of an option price:
    vega = (1 / np.sqrt(2 * np.pi)) * S * np.sqrt(T) * np.exp(-(si.norm.cdf(d1, 0.0, 1.0) ** 2) * 0.5)

    #Newton's Method :
    epsilon = 0.000001
    x0 = sigma_init
    xnew  = x0
    xold = x0 - 1

    while abs(xnew - xold) > epsilon:

        xold = xnew
        xnew = (xnew - fx - V0) / vega

        return abs(xnew)
```

This function will return the implied volatility for a given bunch of inputs.

Before implementing the function, we calculated the strike price for each option by using the following formula: $S = ITM * F/D(0,T)$

RESULTS

We obtain the following board which gives for each option and expiry the resulted implied volatility:

expiry	fwrd\itm	0,7	0,8	0,9	1	1,1	1,2	1,3
5	107,19303	54,30%	35,90%	15,57%	2,45%	0,07%	0,03%	0,03%
7	106,871933	45,49%	29,93%	11,46%	2,39%	0,15%	0,02%	0,02%
10	106,392089	37,56%	24,36%	7,68%	2,31%	0,26%	0,00%	0,02%

PROJECT PHASE 2: VOLATILITY INTERPOLATION

PROBLEMATIZATION

The idea is basically to try the linear, quadratic, and cubic interpolation scheme in the strike direction for *ITM* = 0.75, 0.85, 0.95, 1.05, 1.15, 1.25 for each expiry.
Then we will do the same interpolation schemes in the expiry direction for ITM = 0.7, 0.8, 0.9, 1.1, 1.2, 1.3 for expiries 3 months, 6 years and 9 years.

METHODOLOGY

To perform the different interpolation scheme we use the Python package called interpolate.interp1d from the library spicy which is used to Interpolate a 1 dimension function which array type x and y inputs.
We calculated the volatility approximation in the strike direction for the new set of ITM value for each expiry by using the following Python calculation:

Different interpolation schemes in the strike direction for ITM = 0.75, 0.85, 0.95, 1.05, 1.15, 1.25 for each expiry.

```
itm=df2.itm.unique()
expiry = df2["expiry"].unique()
itm_new = [0.75,0.85,0.95,1.05,1.15,1.25]
```

```
for i in range(0,len(expiry)) :
    globals()['expiry_{0}'.format(i)] = df2.loc[df2.expiry == expiry[i]]
    f_linear = interpolate.interp1d(globals()['expiry_{0}'.format(i)].itm, globals()['expiry_{0}'.format(i)].sigma ,kind='sline
    f_quadratic = interpolate.interp1d(globals()['expiry_{0}'.format(i)].itm,globals()['expiry_{0}'.format(i)].sigma ,kind='qua
    f_cubic = interpolate.interp1d(globals()['expiry_{0}'.format(i)].itm,globals()['expiry_{0}'.format(i)].sigma ,kind='cubic')
    ylinear = f_linear(itm_new)
    yquadratic = f_quadratic(itm_new)
    ycubic = f_cubic(itm_new)
    globals()['vol_{0}'.format(i)]= pd.DataFrame(list(zip(globals()['expiry_{0}'.format(i)].expiry,itm_new,ylinear,yquadratic,y
                columns =['expiry','itm_new','linear', 'quadratic','cubic'])
```

We calculated the volatility approximation in the strike direction for the new set of expiry value for each ITM by using the following Python calculation:

interpolation schemes in the expiry direction for ITM = 0.7, 0.8, 0.9, 1.1, 1.2, 1.3 for expiries 3 months, 6 years and 9 years

```
: expiry_new = [90/365,6,9]
```

```
for i in range(0,len(itm)) :
    globals()['itm_{0}'.format(i)] = df2.loc[df2.itm == itm[i]]
    f_linear = interpolate.interp1d(globals()['itm_{0}'.format(i)].expiry, globals()['itm_{0}'.format(i)].sigma ,
                                    kind='slinear')
    f_quadratic = interpolate.interp1d(globals()['itm_{0}'.format(i)].expiry,globals()['itm_{0}'.format(i)].sigma ,
                                       kind='quadratic')
    f_cubic = interpolate.interp1d(globals()['itm_{0}'.format(i)].expiry,globals()['itm_{0}'.format(i)].sigma ,
                                   kind='cubic')
    ylinear = f_linear(expiry_new)
    yquadratic = f_quadratic(expiry_new)
    ycubic = f_cubic(expiry_new)
    globals()['volex_{0}'.format(i)]= pd.DataFrame(list(zip(globals()['itm_{0}'.format(i)].itm,
                                                            expiry_new,ylinear,yquadratic,ycubic)),
                columns =['itm','expiry_new','linear', 'quadratic','cubic'])
```

RESULTS

- Linear, quadratic and cubic approximation scheme in the strike direction for *ITM* = 0.75, 0.85, 0.95, 1.05, 1.15, 1.25 for each expiry.

Linear spline

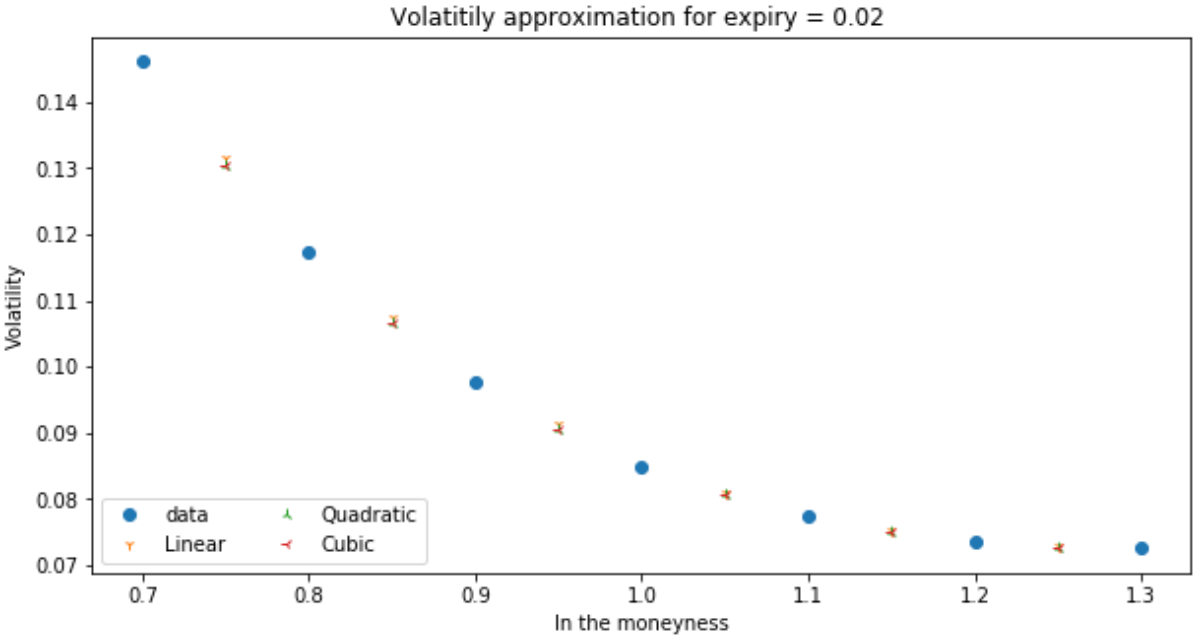
Expiry\ ITM	0,75	0,85	0,95	1,05	1,15	1,25
0,0	13,17%	10,75%	9,14%	8,11%	7,54%	7,30%
0,0	12,76%	10,58%	9,13%	8,21%	7,69%	7,47%
0,1	12,39%	10,46%	9,16%	8,35%	7,88%	7,69%
0,5	11,98%	10,28%	9,15%	8,44%	8,03%	7,87%
1	11,58%	10,13%	9,16%	8,55%	8,20%	8,06%
2	11,18%	9,97%	9,16%	8,65%	8,36%	8,24%
3	10,79%	9,82%	9,17%	8,76%	8,53%	8,44%
4	10,37%	9,64%	9,16%	8,85%	8,68%	8,61%
5	9,97%	9,49%	9,16%	8,96%	8,84%	8,80%
7	9,57%	9,33%	9,17%	9,07%	9,01%	8,99%
10	9,14%	9,14%	9,14%	9,14%	9,14%	9,14%

Quadratic spline

Expiry\ ITM	0,75	0,85	0,95	1,05	1,15	1,25
0,0	13,05%	10,65%	9,07%	8,06%	7,49%	7,26%
0,0	12,65%	10,49%	9,06%	8,16%	7,65%	7,44%
0,1	12,29%	10,37%	9,11%	8,30%	7,85%	7,66%
0,5	11,89%	10,21%	9,10%	8,40%	8,00%	7,84%
1	11,51%	10,07%	9,12%	8,52%	8,18%	8,04%
2	11,11%	9,92%	9,12%	8,62%	8,34%	8,22%
3	10,74%	9,78%	9,15%	8,74%	8,52%	8,42%
4	10,33%	9,61%	9,14%	8,84%	8,67%	8,60%
5	9,95%	9,47%	9,15%	8,95%	8,84%	8,79%
7	9,56%	9,32%	9,16%	9,06%	9,01%	8,98%
10	9,14%	9,14%	9,14%	9,14%	9,14%	9,14%

Cubic spline

Expiry\ ITM	0,75	0,85	0,95	1,05	1,15	1,25
0,0	13,04%	10,65%	9,07%	8,06%	7,49%	7,26%
0,0	12,64%	10,49%	9,06%	8,16%	7,65%	7,44%
0,1	12,28%	10,38%	9,11%	8,30%	7,85%	7,67%
0,5	11,88%	10,21%	9,10%	8,40%	8,00%	7,84%
1	11,50%	10,07%	9,12%	8,52%	8,18%	8,04%
2	11,11%	9,92%	9,12%	8,62%	8,34%	8,22%
3	10,73%	9,78%	9,15%	8,74%	8,52%	8,42%
4	10,33%	9,61%	9,14%	8,84%	8,67%	8,60%
5	9,94%	9,47%	9,15%	8,95%	8,84%	8,79%
7	9,56%	9,32%	9,16%	9,06%	9,01%	8,98%
10	9,14%	9,14%	9,14%	9,14%	9,14%	9,14%



- Linear, quadratic and cubic approximation scheme in the strike direction for ITM = 0.7, 0.8, 0.9, 1.1, 1.2, 1.3 for expiries 3 months, 6 years and 9 years.

Linear spline

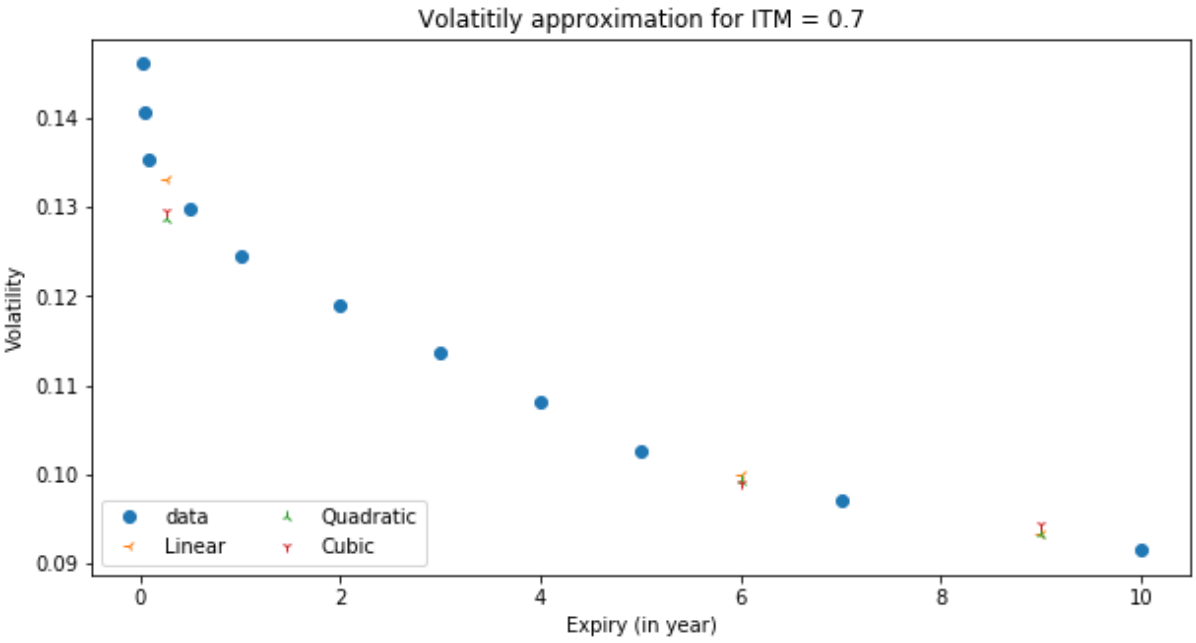
itm / expiry (in years)	0,25	6	9
0,7	13,32%	9,99%	9,33%
0,8	11,13%	9,56%	9,24%
0,9	9,65%	9,26%	9,17%
1	8,68%	9,07%	9,13%
1,1	8,09%	8,96%	9,11%
1,2	7,80%	8,90%	9,09%
1,3	7,73%	8,88%	9,09%

Quadratic spline

itm / expiry (in years)	0,25	6	9
0,7	12,87%	9,93%	9,33%
0,8	10,95%	9,53%	9,24%
0,9	9,66%	9,26%	9,18%
1	8,81%	9,08%	9,14%
1,1	8,29%	8,97%	9,11%
1,2	8,04%	8,92%	9,10%
1,3	7,98%	8,91%	9,10%

Cubic spline

itm / expiry (in years)	0,25	6	9
0,7	12,96%	9,91%	9,43%
0,8	11,00%	9,52%	9,29%
0,9	9,67%	9,26%	9,19%
1	8,80%	9,08%	9,12%
1,1	8,28%	8,98%	9,08%
1,2	8,01%	8,93%	9,06%
1,3	7,95%	8,92%	9,06%



SOURCES:

Implied Volatility Calculations with Python By Aaron Schlegel:
https://www.math.ubc.ca/~pwalls/math-python/roots-optimization/newton/

Mathematical python: Newton's algorithm https://www.math.ubc.ca/~pwalls/math-python/roots-optimization/newton/

scipy.interpolate.interp1d reference guide:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html