Project Report of FE5116

1. Implementation of Functions

1.1 Black formula

This function just implements BS formula without considering discount factor. What should pay attention is that when $k \leq 0 \; and \; k \to \infty$

Thus when $k \leq 0$, u = forward price

When $k \to \infty$, u=0

Some details: why this function considered the situation $k < 0$ rather than giving error message because this function is used to compute pdf of S(T) subsequently, it has a term f(x-h), and f is BS formula. When x gives a very small value like 1e-7, (x-h) may become negative. Thus, I need the price even k becomes negative.

1.2 Interest rate interpolation

Bootstrap method:

$$r(t,T) = \frac{log\left(\frac{df_t}{df_T}\right)}{T-t}$$

The discount factor in time t and T are known, the interest rate from t to T can be obtained via bootstrapping. For t = 0, $df_t = 1$.

If there is an array of size N discount factors and tenors, vector calculation is implemented to obtain the interest rate (**there is no loop**).

Some details:

MakeDepoCurve(ts,dfs)

- Piecewise constant method interpolates constant interest rate from t to T.
- Only can extrapolate for 30 days. if you give a t which in [max tenor, max tenor+30/365], it will give you the same interest rate as t = max tenor
- Error message generated if input tenor exceeds the max tenor
- **Return Type: function handle**

  This function handle can accept a tenor and return the interest rate after bootstrap method. Pay attention that bootstrap method will only **execute once**, what the function do is to use **binary search** to get which interval $[T_i, T_{i+1}]$ should be according to the variable t you give.

getRateIntegral(curve,t)

- Considering the interest rate is piecewise, just use **Rectangle Rule** will be ok for the integration.
- yi = arrayfun(curve, [0:N-1]*delta), use **arrayfun** to generate all the points we need.

1.3 Forward spot

$$S(t) = F_t(t+\tau) = X_t e^{\int_t^{t+\tau}[r(u)-y(u)]du}$$

$$G_t(T) = F_t(T+\tau) = X_t e^{\int_t^{T+\tau}[r(u)-y(u)]du}$$

As soon as we have two curves for domestic market and foreign market. We can compute forward price use the formula above.

Pay attention that what we get from market is S_t not X_t because X_t is not observable from market.

Some details:

makeFwdCurve (domCurve , forCurve , spot , tau)

- **Return type: function handle:**

  This function will give two output, spot price and interest rate difference $r(u) - y(u)$ which are useful to integrate.
- When the t exceeds max tenor, it will give error message.

fwdSpot = getFwdSpot (curve , T)

- **Considering we already has S_t thus we only integrate from τ to $T+\tau$.**
- Considering both curves are Piecewise constant, I still use **Rectangle Rule.**
- Give error message if input T exceed max tenor

1.4 Conversion of deltas to strikes

According the type and delta of a option, we can easily get N(d$_1$), under the situation we know S, T, sigma, strike becomes the only variable of N(d$_1$). Thus, we can use root search method to find K.

getStrikeFromDelta (fwd , T, cp , sigma , delta)

- Use secant method to find root, however, this method is not stable.
- Noticed that if we know the value of N(d_1), we can use the function norminv() to get the value of d_1, then according to the definition of d_1, we can get the strike price

1.5 Interpolation of implied volatility in strike direction

makeSmile (fwdCurve , T, cps , deltas , vols)

- Use f**wdCurve** to get the forward price
- Considering at a fixed T, we have a vector of volatilities. so we use **getStrikeFromDelta** to get the strike price according to the volatility
- Use **getBlackCall** to calculate the call price
- Now we have a vector of strikes against a vector of call prices.
- Check three types of arbitrage chances. And give error message if arbitrage exists.
- Use spline function get a structure **polyfit = spline(strikes,vols)**
- Calculate the parameters with the following order:

$$K_L = K_1 * \frac{K_1}{K_2}, K_R = \frac{K_N * K_N}{K_{N-1}}$$

  According $0.5 = \tanh^2\big(b_R(K_R - K_N)\big)$, $0.5 = \tanh^2\big(b_L(K_1 - K_L)\big)$ we can get b_R, b_L

  Then according $a_R b_R = \sigma'(K_N), -a_L b_L = \sigma'(K_1)$ we can get a_R, a_L
- How to ask $\sigma'(x)$? pay attention from the spline function, we can get the parameters of the cubic polynomial in every interval and we know in a interval $[a_i, a_{i+1}]$, the cubic polynomial will like this:

$$y = \alpha(x - a_i)^3 + \beta(x - a_i)^2 + \gamma(x - a_i) + \theta$$

  Thus, we can get $\sigma'(K_N)$ and $\sigma'(K_1)$ according to which interval it belongs to.
- **Return type: function handle**

  This function will first judge which interval the input t should be(t>K_N $or\ t < K_1\ or\ K\ in\ (K_1, K_N)$) give volatility according to t.

Pay attention that the interpolation will only execute once.

getSmileVol (curve , Ks)

- Use arrayfun(curve, Ks) to accept vector input of strike.

1.6 Construction of implied volatility surface

The very simple logic about this task is that we already have ten tenors. From every tenor, we can have a smile using spline function to interpolate. After that, if we want to know $\sigma(T,K)$ given any T and K. first we use fwdCurve to get the forward price according to T, then we can get the moneyness value $K/G_0(T)$. Then we should find which interval T should be and get $G_0(T_i)\ and\ G_0(T_{i+1})$.

Then according to

$$\frac{K}{G_0(T)} = \frac{K_i}{G_0(T_i)} = \frac{K_{i+1}}{G_0(T_{i+1})}$$

We can get K_i, K_{I+1}. Then we can get $\sigma_i(K_i)$,$\sigma_{i+1}(K_{i+1})$ according to the smile which belongs to $T_i\ and\ T_{i+1}$. Finally, we use the linear variance interpolation along moneyness lines to get $\sigma(t,T)$.

makeVolSurface (fwdCurve , Ts , cps , deltas , vols)

- **Return Type:** struct, contains ten smile functions and tenor and fwdCurve information
- Check arbitrage for k = fwd in different tenor and give error message if there exists arbitrage chance.

getVol (volSurf , T, Ks)

- Considering a given T, use binary search to find T_i, T_{i+1},
- Considering the moneyness line get $\sigma_i(K_i)$,$\sigma_{i+1}(K_{i+1})$
- Use linear variance interpolation to get $\sigma(t,T)$

1.7 Compute the probability density function

Use Taylor expand we can get the second derivative can use the following expression:

$$f(x)'' = \frac{f(x+h) + f(x-h) - 2 * f(x)}{h^2}$$

getPdf (volSurf , T, Ks)

- Get volatility and forward price according to the given parameters volsurf, T and Ks.
- Calculate the call option price
- Make above as a single function f(k), which you input a strike and return the call option price.
- Use the formula above to calculate second derivative, set h = 1e-7.
- Already considered vector input Ks

1.8 Compute forward prices of European options

Just execute the integration:

$$V = \int_0^\infty f(x)\phi(x)dx$$

getEuropean (volSurface , T, payoff , ints)

- Use nargin to control value of ints
- If nargin==3, use default setting with interval = 0.1

  Define: **Func = @(x)pdf(x).*payoff(x)**
- If nargin==4, the interval of integration should be set. Notice that we are integral zero to infinite, if we use Rectangle Rule, it's hard for us to find the upper limit of the integration.

  However, with a very big x, $f(x)$ will tend to be zero, thus a tolerable error is set, using the following logic to find the upper limit:

```
e = 1e-8;
b = 0.1;
while pdf(b+1)>e
    b=b+1;
end
```

In this way we can find the upper limit b and use rectangle rule to do the integration.

- **Notice that the smaller interval you set, the more time it will spend.** If the 'ints' parameters are not used and we directly use MATLAB function integral, the speed will be improved significantly.

2. Illustration of Function Output

2.1 Forward Curve

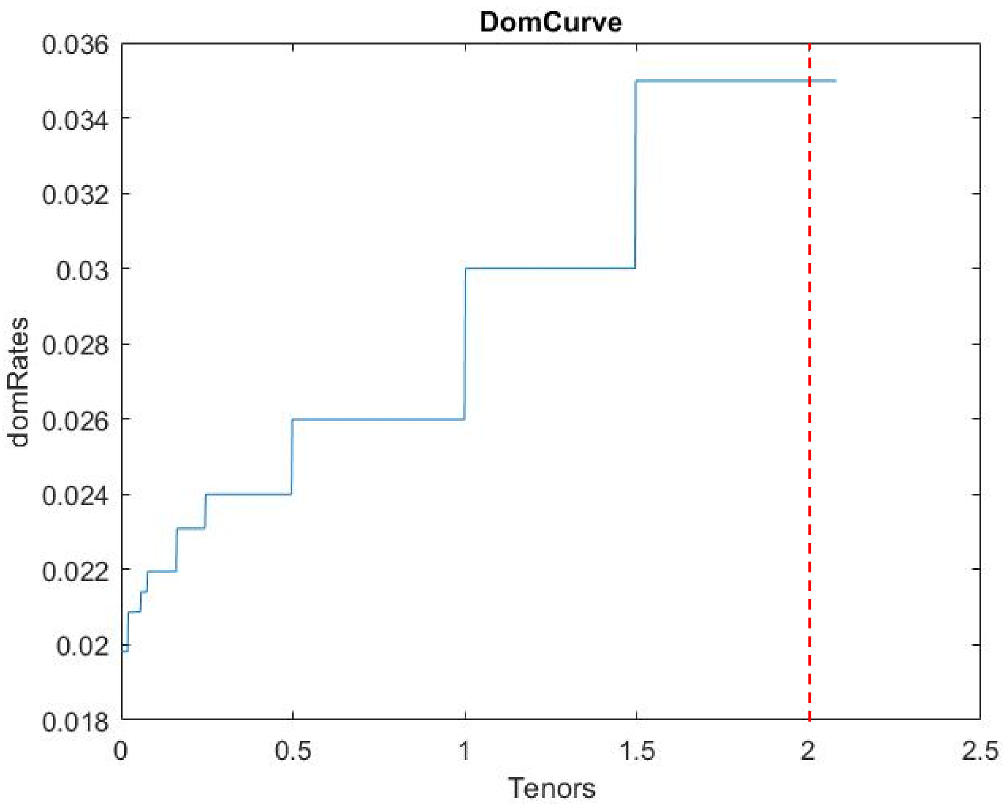

Figure 1 Interpolation of domCurve

In Figure 1, the rates are constant between each tenor after implementing piecewise constant interpolation. For time beyond the last tenor (red line) within 30 days, extrapolation is done with value equals to the last rate. If time exceeds 30 days after the last tenor, there will be no extrapolation. These values will be used to generate forward curve shown in Figure 2.

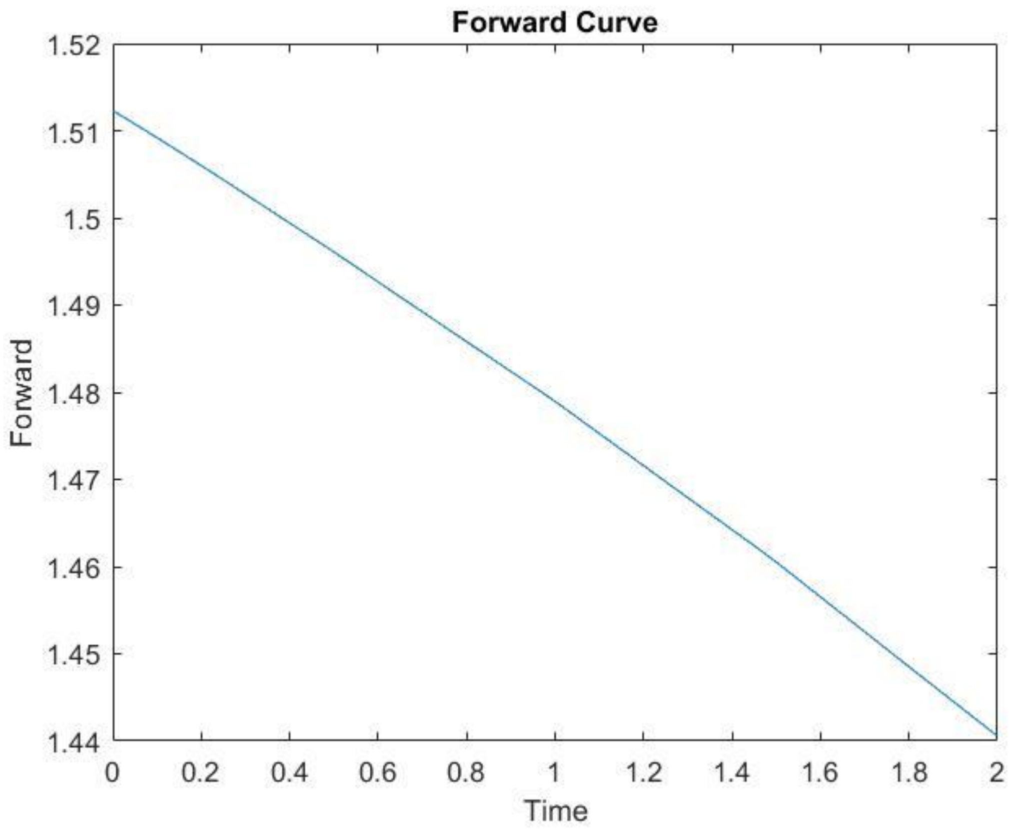

Figure 2 Forward Curve

As shown in Figure 2, the forward curve is plotted with input from T = 0 to 2 at an interval of 0.04 with getFwdSpot function. The forward curve is smooth and starts with 1.5123 at T = 0. As T increases, the forward spot rate decreases since the difference between domCurve and forCurve is negative.

2.2 Volatility Smile

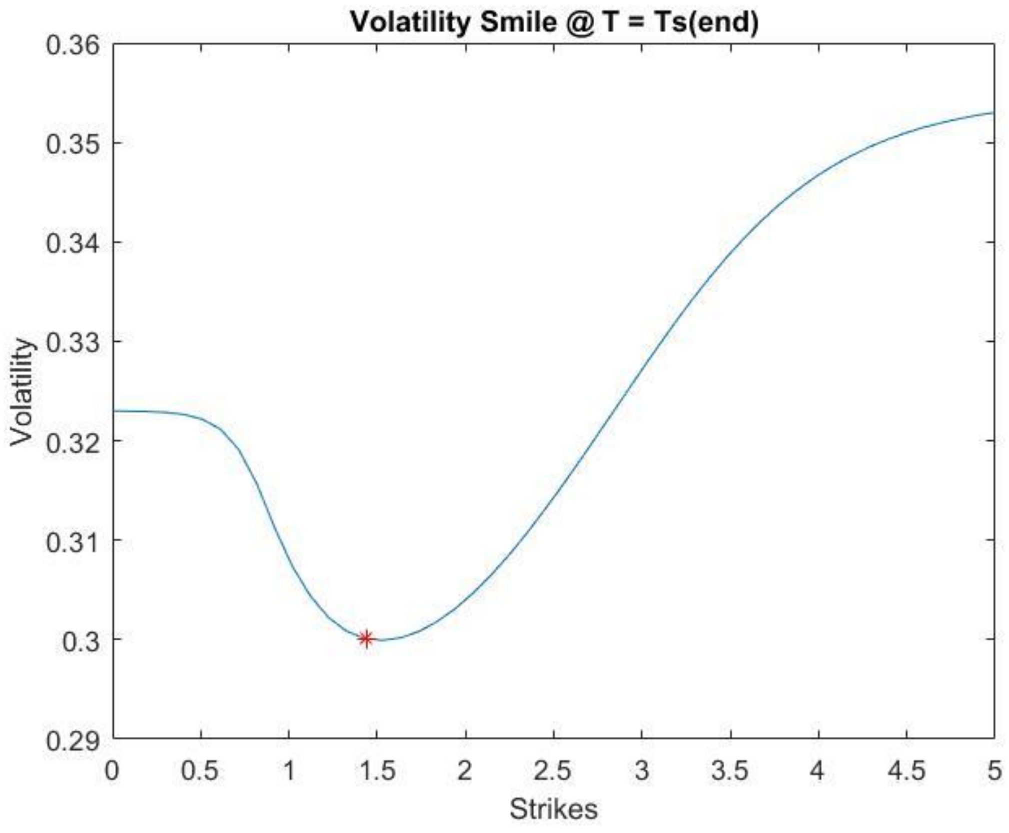

Figure 3 Volatility smile at T = 2 with K = 0 to 5

Function getVol generates volatilities of an array of strikes Ks at a specific time T using volSurface obtained from the makeVolSurface function. Figure 3 shows the results of getVol. At T = 2, the corresponding volatilities with K = 0 to 5 at an interval of 0.1 is plotted. As illustrated in Figure 3, the interpolation and extrapolation are smooth, especially at the joints. The red mark on the graph is the atmfvol = 0.3001 when K equals to forward spot at T = 2.

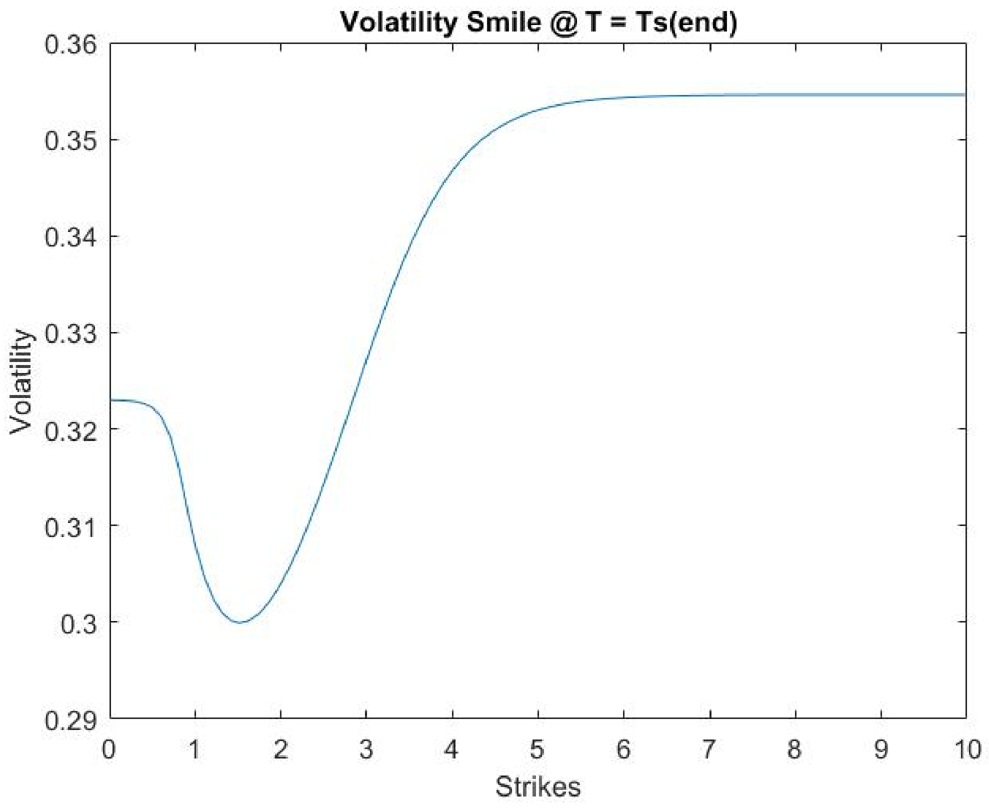

Figure 4 Volatility Smile at T = Ts(end) with K = 0 to 10

Figure 4 shows that the when K is far larger than the given interval, from K = 5 to 10, the volatility value is smooth and gently flattened.

2.3 Volatility Surface

Function getVol generates volatility at any time and strike from the volatility surface created by makeVolSurface function. Therefore, by plotting strike from 0 to 5 and an interval of 0.1 against time from 0 to 2 at an interval of 0.04, a volatility surface (Figure 5) is obtained.

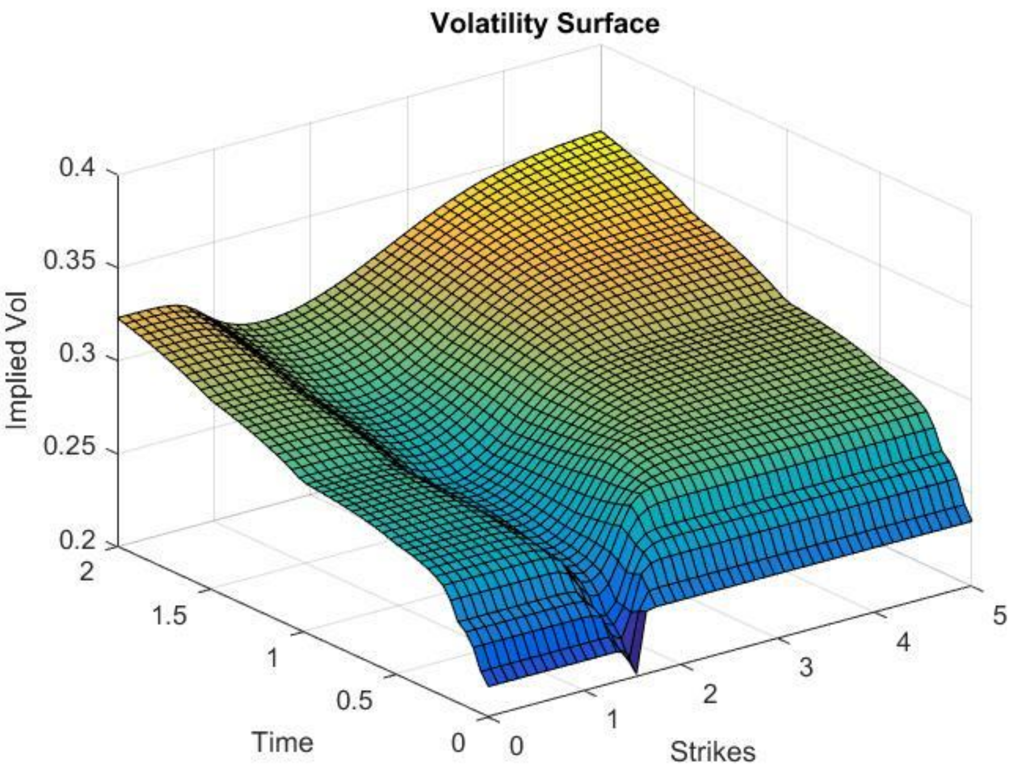

Figure 5 Volatility Surface

The surface is overall smooth at each expiry date and strike. The result is the same as expected. There is a volatility smile at each expiry date, while along the time axes, implied volatility increases as time increases due to greater uncertainty for longer tenors.

2.4 Pdf Curve

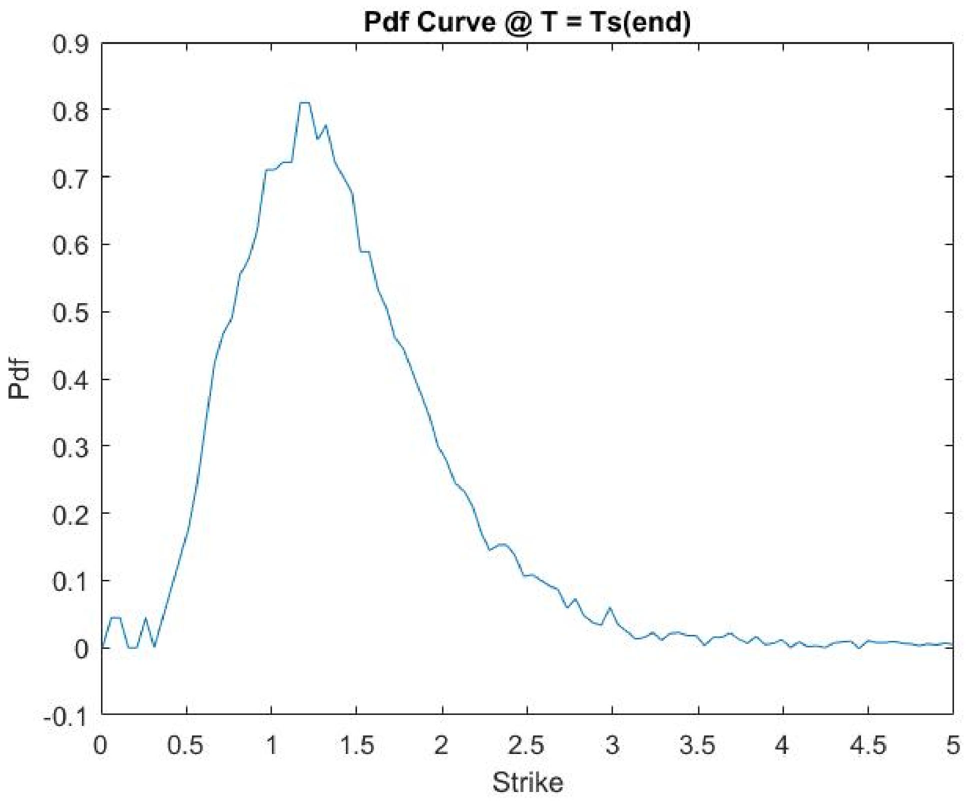

Figure 6 Pdf Curve

Figure 6 demonstrates the Pdf curve at various strikes from 0 to 5 at expiry time equals the last tenor which is 2. The curve is generally smooth. The fluctuations along the curve may be caused by discontinuities in the second derivatives of call option prices.

3. Tests

We write tests for all getXXX functions, and tests are illustrated as follows

3.1 getBlackCall

Test whether the function return correct option price.
If difference between calculated value and known option price is less than threshold (0.001 in our test), we take the function as correct.

3.2 getRateIntegral

Test whether integral of local rate can match with discount factor, and whether second derivative of integral is zero.
Firstly, we check

$$e^{-\int_0^t r(u)du} = M_t.$$

Then, considering that the curve is stepwise constant, the derivative of output of getRateIntegral should be constant. Therefore, if the second order difference between integral is less than threshold, we take the difference as 0 and the function as correct.

3.3 getFwdSpot

Test whether forward spot is correct, and whether the function output is continuous.
Firstly, we check

$$G(T) = S_t e^{\int_0^T [r(u)-y(u)]du} = S_t * \frac{N_t}{M_t}.$$

Then, the function should be continuous and nearly linear. So similarly, we check the second order difference.

3.4 getStrikeFromDelta

Test whether we can get right deltas through calculated strikes.
From this function, we derive strikes, from which we calculate new deltas. And we check whether new deltas match with original ones.

3.5 getSmileVol and getVol

Test whether calculated vols for certain tenor are same as the raw known vols.

With provided tenor, we derive vols from function, and check whether they match with vols from market data.

3.6 getPdf

Test whether ingetral of calculated pdf is 1, and whether expectation under such pdf is the forward.

$$\int_0^\infty pdf_t(x)dx = 1, \int_0^\infty x * pdf_t(x)dx = G(t).$$

We check the gap between integral of pdf and 1, and gap between integral of pdf times x and forward price. The function passes test if both gaps are less then threshold.

3.7 getEurpean

Test whether European call and put option prices are same as analytical results and can satisfy call-put parity, and whether non vanilla payoff can be approximated by linear combination of vanilla options. Furthermore, test whether a nonvanilla payoff can be approximated with a linear combination of vanilla options.

$$C(K) - P(K) = G(T) - K$$

For the nonvanilla payoff, we construct a simple test example:

$$\max(S_T - K) - \max\big(S_T - (K + \mathit{\Delta} K)\big).$$

And check whether option price with this payoff is same as call option price with strike K minus call option price with strike $K + \Delta K$.

Word Count (1995)

Individual Reflections

Chen Xilin

I implemented getBlackCall_test, getFwdSpot_test, getRateIntegral_test, PutCallParity, getBlackPut, validated getBlackCall, makeDepoCurve, getRateIntegral, makeFwdCurve, and getFwdSpot, and reviewed the rest of test functions. I wrote getBlackCall, makeDepoCurve, getRateIntegral by myself as well.

Regarding makeDepoCurve and getRateIntegral the thing to take note is the constrain to generate error message if you give a tenor that exceeds the max tenor, I validated and optimized the constrains. Since the curve is stepwise constant, the test file validates this feature by calculating the derivative of the integral. What need to mention is that when applying loop to calculate derivative, the smallest unit of stepwise length is 0.001, since 0.001 is the step interval when implementing getRateIntegral.

For getFwdSpot, the tricky thing is that what we get from market is S_t not X_t because X_t is not observable from market. And again the curve is stepwise constant and fwdCurve is constant in the case that domCurve is the same as forCurve. And the continuity of forward spot function is tested.

Li Xin

I did the test part with my teammates. We discussed ideas and details for the test together and I wrote the report of this part. With a weak programming background, I found the project is very difficult for me. But luckily, my teammates are very excellent and skilled at MATLAB and they are very generous to share. The learning process itself is a great journey for me. By testing and replicating their codes, I made great advance in MATLAB, and sort through the logic and the implementation details of the project, some of which might be ignored by myself. For example, in getBlackCall function, it is necessary to consider the situation k<0 to make the pdf function of St following right. By meeting and discussion, my teammates patiently explained my confusion and pointed out where I was doing wrong. Good cooperation makes my team very efficient. I appreciate what I've learned from the project and my teammates.

Shan Changhan

I mainly focus on the test part. I code tests for functions of strikes, volatility, probability distribution functions and European options. And improve efficiency of tests for functions of forward price and integration of rates.

One problem I encounter is low accuracy of our results when they are calculated from primitive integration method, i.e. sum the value of small intervals times interval length. It mainly rises from the unstable performance of pdf near 0. So I avoid involving pdf of very small x. Anyway, the result from integration is not exactly the same as an analytical one. I relax the threshold of judging whether two values are same and take epsilon as 0.001 instead of 0.000001.

Besides, MATLAB provides some integration functions, but because of different installed versions of my teammates, these functions do not perform consistently.

Zheng Hao

I mainly focus on the whole project

Implement functions:

getBlackCall, makeDepoCurve, getRateIntegral, makeFwdCurve, fwdSpot, getStrikeFromDelta, makeSmile, getSmileVol, makeVolSurface, getVol, getpdf, getEuropean, BinarySearch, Secant.

Following the steps given in your instruction, I wrote and implemented all the functions.

Issues:

1. getBlackCall may receive negative strike as input because when I calculate second derivative, the (x-h) in f(x-h) may be negative. Thus, when k<=0, I set value as fwd.
2. when I calculate the forward price, noticed that the given spot price is not X_0,
3. when I use root search to find strike price, I found secant method is not stable. Instead, I change norminv() to directly solve strike price

   when I do the spline interpolation, I was wondering how to calculate the first derivative of edge point, noticed that the spline expression should be

$$y = \alpha(x - a_i)^3 + \beta(x - a_i)^2 + \gamma(x - a_i) + \theta$$

   Then I get the result I want

4. when I use getpdf, noticed that when k becomes very small, the result is not very accurate. Thus, I make the ‘h’ in f(x+h) to be 1e-7, which is enough to handle most case.
5. When I consider how to integrate 0 to inf in getEuropean, I noticed that when x becomes large, pdf(x) will be zero, thus I set a upper limit for the integration, when pdf(b)<error, I will just integrate from 0 to b.
6. Happy experience about the project. Really learned a lot.

Zhou Lingjin

I mainly focus on the whole project

Implement functions:

getBlackCall, makeDepoCurve, getRateIntegral, makeFwdCurve, fwdSpot, getStrikeFromDelta, makeSmile, getSmileVol, makeVolSurface, getVol, getpdf, getEuropean, BinarySearch, Bisection, plotVolSurface.

Following the steps given in your instruction, I wrote and implemented all the functions. Instead of using arrayfun, I used loops to solve the problems. With different methods, my answer is used as a cross references for Zheng Hao’s answer to ensure the correctness of our results. I implemented binarysearch for the right bucket and piecewise constant interpolation in getRateIntegral. Bisection was used in getStrikeFromDelta to search for the corresponding K from Delta using Black Scholes formula. In addition, I plotted the results using visualizations tools to better illustrate the output of the function output. Last but not least, I help to compile this report.

Issues:

1. In makeSmile function, I made great effort to search for the suitable function to find the first derivative of interpolation. Eventually, I used p_der=fnder(pp,1).
2. When smoothen the joints between interpolation and extrapolation in makeSmile, I understood the given formula wrongly and caused discontinuities in the joints. I checked with the formula carefully and corrected it.

3. This project is challenging but meaningful to me. It gives me a chance to calculate and plot the volatility surface on my own which allows me to have a better understanding of option pricing.