\_

**Project report**

**Describe clearly the implementation of each of your functions.**

**1.1 Black formula**

This function just implements BS formula without considering discount factor. What should pay attention is that when

Thus when , u = forward price

When , u=0

**Some details:** why this function considered the situation rather than giving error message because later I will use this function to compute pdf of S(T), 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:

As we know the discount factor in time t and T, we can bootstrap the interest rate from t to T.

And for t = 0, . Thus, if I get an array of size N discount factors and an array of size N times, I can use vector calculation to get the interest rate (**there is no loop**).

**Some details:**

**MakeDepoCurve(ts,dfs)**

* This the piecewise method which means the interest rate from t to T is a constant.
* 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
* It will generate error message if you give a tenor exceed 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 ] 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.

2.3 **Forward spot**

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 not because 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 which are useful to integrate.

* When the t exceed max tenor, it will give error message.

**fwdSpot = getFwdSpot (curve , T)**

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

**2.4 Conversion of deltas to strikes**

According the type and delta of a option, we can easily get N(d1), under the situation we know S, T, sigma, strike becomes the only variable of N(d1). 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(d1), we can use the function norminv() to get the value of , then according to the definition of , we can get the strike price

**2.5 Interpolation of implied volatility in strike direction**

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

**process:**

* 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:

According we can get

Then according we can get

* How to ask ？pay attention from the spline function, we can get the parameters of the cubic polynomial in every interval and we know in a interval [], the cubic polynomial will like this:

Thus, we can get and according to which interval it belongs to.

* **Return type: function handle**

This function will first judge which interval the input t should be(t>) 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.

**2.6 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 given any T and K. first we use fwdCurve to get the forward price according to T, then we can get the moneyness value . Then we should find which interval T should be, and get .

Then according to

We can get . Then we can get , according to the smile which belongs to . Finally we use the linear variance interpolation along moneyness lines to get .

**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 ,
* Considering the moneyness line get ,
* Use linear variance interpolation to get

**2.7 Compute the probability density function**

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

**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 you the call option price.
* Use the formula above to calculate second derivative, I set h = 1e-7.
* Already considered vector input Ks

**2.8 Compute forward prices of European options**

Just execute the integration:

**getEuropean ( volSurface , T, payoff , ints )**

* Use nargin to just whether we use ints
* If nargin==3, we use default setting with interval = 0.1

Define: **Func = @(x)pdf(x).\*payoff(x)**

* If nargin==4, then we should consider the interval when we integrate. 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, I find that for a very big x, will tend to be zero, thus I set a tolerable error, use 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.** However, if we don’t use the ‘ints’ parameters and directly use matlab function integral, the speed will obviously improve.

\_ 2. Illustrate with charts the output of your get\* functions, demonstrating their output is

smooth and behaves as expected. When possible shows in the same chart also analytical

results or input marks.

**2.1 Forward Curve**

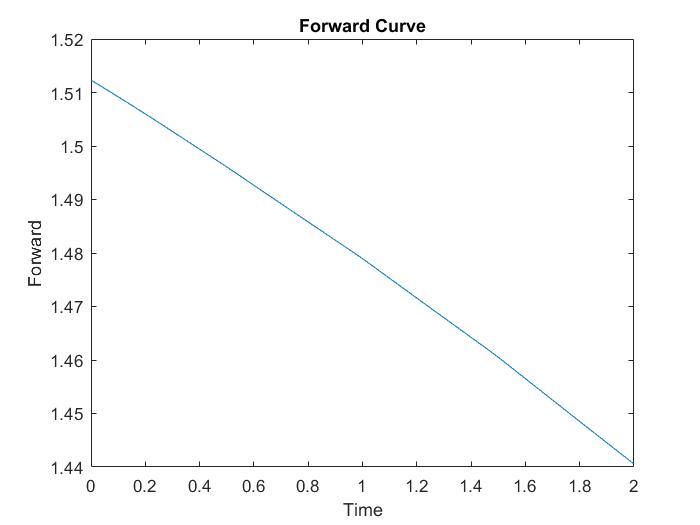


Figure 1 Forward Curve

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

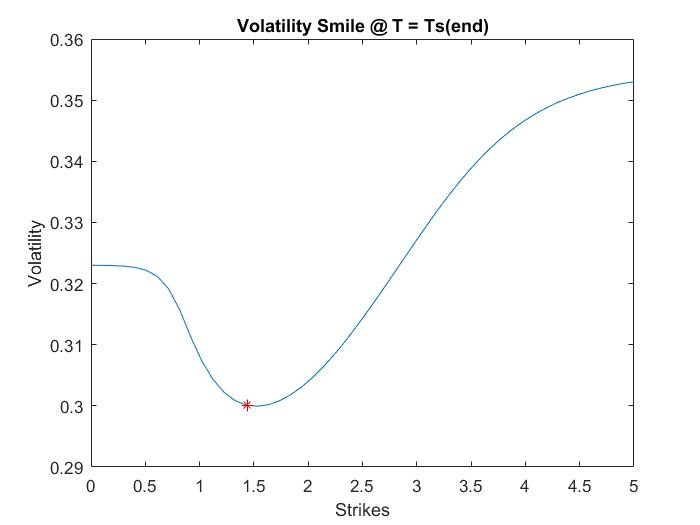


Figure 2 Volatility smile at T = Ts(end) with K = 0 to 5

Function getVol generates volatilities of an array of strikes Ks at a specific time T using volSurface obtained from the makeVolSurface function. Figure 2 shows the results of getVol. At T = Ts(end) (T = 2), the corresponding volatilities with K = 0 o 5 at an interval of 0.1 is plotted. As illustrated in Figure 2, 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.

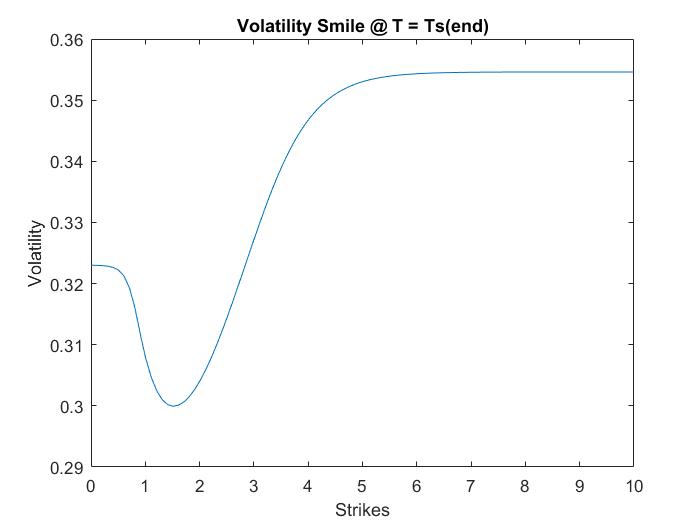


Figure 3 Volatility Smile at T = Ts(end) with K = 0 to 10

Figure 3 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 4) is obtained.

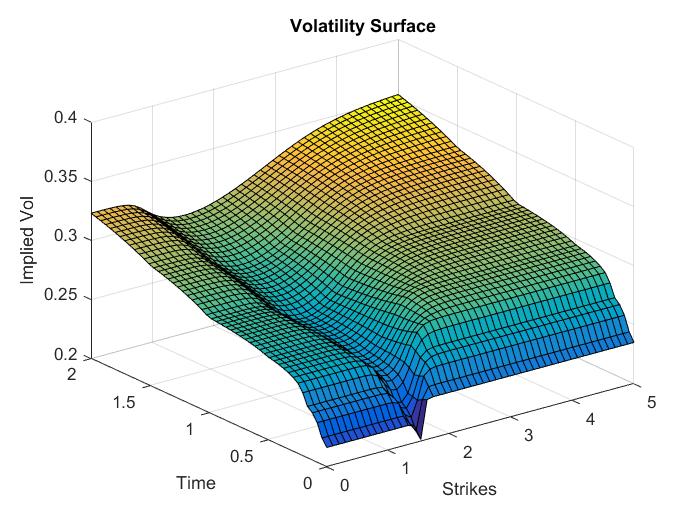


Figure 4 Volatility Surface

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

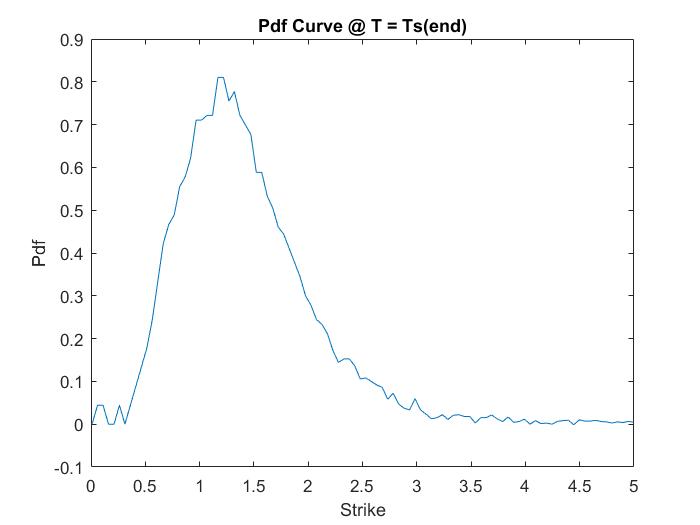


Figure 5 Pdf Curve

Figure 5 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 ???

**Tests**

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

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.

1. getRateIntegral

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

First we check

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

1. getFwdSpot

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

First we check

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

1. 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.

1. 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.

1. getPdf

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

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.

1. 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.

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

And check whether option price with this payoff is same as call option price with strike minus call option price with strike .

Reflections (individual)

\_ List what functions you implemented.

\_ Describe what issues you faced and how you addressed them.

\_ Maximum 300 words

1. 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.