**2.2 Interest rate interpolation**

**2.2.1. makeDepoCurve**

**1. Implementation**

With the assumption that interest rates are constant between and , this function calculate the interest rate to make Depocurve based on the formula bellows.

But there is a problem on the part, it will cause the precision problem on the calculation of , and therefore the getRateIntegral function can’t return the original point. We then modify the formula as follows:

It solves the precision problem successfully, because it has changed the division method to a minus sign out of the Ln function, which improves the precision.

We also calculate rate integral and forward interest rate as follows, which reduces calculations.

**2. Description of Depocurve**

The Depocurve is a struct containing four fields, ir, ts, integ and fwdir. It represents the term structure of interest rates for a country, and it is used for calculating the interest rate on a special time t.

Curve.ir = , It is the interest rate timeseries data.

Curve.ts = It is the times to settlement in years.

Curve.integ = , It reduces calculation in getRateIntegral.

Curve.fwdir = , It save the forward interest rate between and , which can also reduce repeated calculation in getRateIntegral.

**2.2.1. getRateIntegral**

**1. Implementation**

For , we assume that the interest rate between 0 and is constant . Then the rate integral should be .

For , let’s assume , then the rate integral should be divided into two part. The first part is the result between 0 and , which is . The second part is the result between and , which is .

For , we assume that the interest rate after is constant . Then the rate integral should be .

**2. Test&Output**

**(1).testInputCheck**

This part checks whether the inputCheck function works successfully. Our test includes the following cases:

1. Curve not complete
2. Curve dimension not match
3. Empty curve
4. Negative or zero value in curve.ts
5. T is not a scalar
6. Negative T
7. T is bigger than the last tenor, more than 30 days

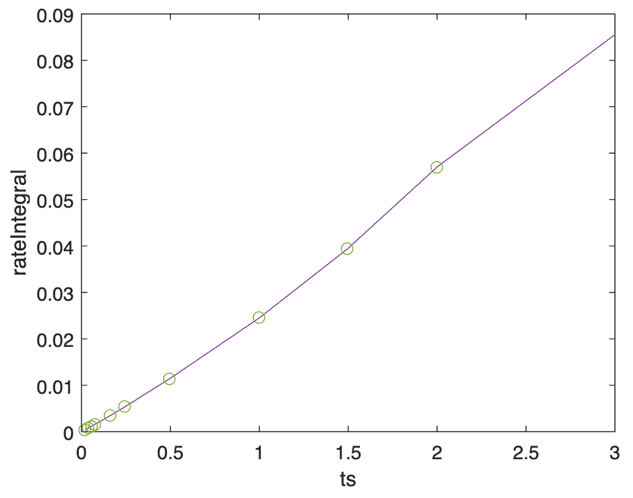
Our function should throw error on all these test cases, or this test fail.

**(2).testOriginalPoint**

For this test, we test whether the original discount factors are equal to the discount factors calculated by the result of getRateIntegral function. The results should be the same, or the test fail.

**(3).testIntegralLinear**

In this test, we get 200 points between each and on equal interval, and test if their rateIntegral are linear and continuous.

****

The dot in this graph are the points in curve.integ. We can see the rate integral returned by getRateIntegral function between and are linear and continuous. But for point , it’s not continuous since the interest rate before and after are different.

**2.3 Forward spot**

**2.2.1. makeFwdCurve**

**1. Implementation**

Firstly, we calculated the cash rate.

Since

Therefore

Then we save the domCurve, forCurve, and in the FwdCurve for the calculation of forward spot price in getFwdSpot function.

**2. Description of FwdCurve**

FwdCurve.domCurve is the domestic IR curve.

FwdCurve.forCurve is the foreign IR curve.

FwdCurve.X0 is the cash rate.

FwdCurve.tau is lag between spot and settlement.

**2.2.1. getFwdSpot**

**1. Implementation**

Since we have saved the cash rate, we apply the formula below to get the forward spot price at any time.

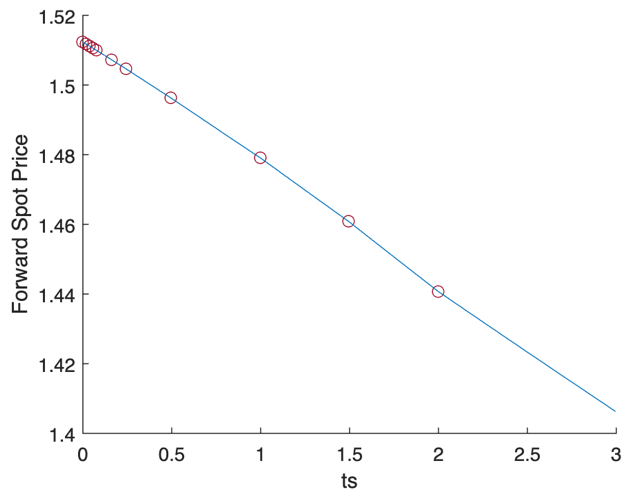
**2. Test&Output**

**(1).testInputCheck**

The test cases of this part are similar to the test case for getRateIntegral. In addition, we also need to check the X0 and tau in FwdCurve.

**(2).testIntegralLinear**

In this test, we get 200 points between each and on equal interval, and test if their forward spot price are linear and continuous.

****

The dot in this graph are the points in fwdCurve.fwd, and this graph is the result of this test. We can see the forward spot price returned by our getFwdSpot function between and are linear and continuous. But for point , it’s not continuous since the forward interest rate before and after are different.