Practical Quant Review

junyan.xu*

January 14, 2019

Contents

*junyanxu5513@gmail.com

1 Instrument Basics

1.1 Durations

Assume time interval is τ, yield is y and term is k. We have price

$$P = \sum_{i=1}^{k} c \frac{1}{(1+y\tau)^i} + \frac{1}{(1+y\tau)^k} \tag{1}$$

If we take the derivative of P versus y

$$dP = -\frac{1}{1+y\tau} \sum_{i=1}^{k} \frac{c\tau i}{(1+y\tau)^i} + \frac{k\tau}{(1+y\tau)^k} \tag{2}$$

The blue part divided by P is the duration. The red part times the blue part over P is the adjusted duration. Key thing to remember is that adjusted duration have a factor of $\frac{1}{1+y\tau}$

2 Forward Measure

2.1 Intuitive Definition

To start with, recall in the HJM framework we have the discount value of a bond price is a martingale in risk neutral measure. This is true because any tradable assets must be a martingale in risk neutral measure otherwise you have arbitrage opportunity

$$d(D(t)B(t,T)) = -\sigma^* D(t)B(t,T)d\widetilde{W}(t) \tag{3}$$

Then recall the definition of the fundamental theorem that there exists a measure between two tradable assets that those two can form a martingale.

If we further assume the measure is at time T, which is the same expiration time of the zero-coupon bonds, then we have:

$$\frac{V(t)}{B(0,T)} = E^T \frac{V(T)}{B(T,T)} = E^T V(T) \tag{4}$$

$$V(t) = B(0,T)E^T V(T) \tag{5}$$

The idea behind this is that, we cannot find a way using bond to hedge our payoff then get money at any condition. Recall the similar risk neutral pricing formula for any contract. But first, think of risk neutral measure as a measure that you can find a martingale such that it prevents you from being using bank account to make arbitrage

$$V(t) = E^B(V(T)D(T)) \tag{6}$$

Since T, B measures are both defined on same event space. The we can naturally get the transformation of the probability on same event to be

$$\widetilde{P}^T(A) = \frac{1}{B(0,T)} \int_A D(T) d\widetilde{P}^B(A) \tag{7}$$

2.2 Link To Risk Neutral Measure

2.2.1 Non-Model Based

We often hear someone say *forward price is martingale under T measure* (*1 over forward price is martingale under S measure*). Firstly, without proof, this statement can be think of as:

- First forward price is the number of bond you hold to hedge the price at time T. Aka the ratio between the expected payoff of contract vs the bond
- Under T measure, this value should equal to $\frac{V(t)}{B(t,T)}$ at time t
- In fact, the ratio value mentioned in first item should equal to $\frac{V(0)}{B(0,T)}$ at time 0
- According to the Tower Theorem, this second and third items are chained up to force the evolution of f from time 0 to time t, the f is a martingale

More intuitively, you can see using T measure just include/offset the factor of $B(0,T)$ and $D(T)$ in the risk neutral measure.

$$f(0) = \frac{E^B V(T) D(T)}{E^B D(T)} = B(0,T) E^B V(T) D(T) = E^T f(T) \tag{8}$$

2.2.2 Change of Numeraire

Thinking intuitively, T measure is just some drift transformation of normal distribution. Under risk-neutral measure, assume we have two process $S(t)$ and $N(t)$ (both are tradable assets).

$$D(t)S(t) = D(0)S(0) \exp(\int_0^t \sigma(t) d\widetilde{W}(u) - 0.5 \int_0^t \left\|\sigma(t)^2\right\| du) \tag{9}$$

$$D(t)N(t) = D(0)N(0) \exp(\int_0^t \mu(t) d\widetilde{W}(u) - 0.5 \int_0^t \left\|\mu(t)^2\right\| du) \tag{10}$$

So therefore under this risk neutral pricing measure, we can have

$$\frac{S(t)}{N(t)} = \frac{S(0)}{N(0)}\exp(\int_0^t (\sigma(t)-\mu(t))d\widetilde{W}(u) - 0.5\int_0^t (\|\sigma(t)^2\| - \|\mu(t)^2\|)du)$$
$$= \frac{S(0)}{N(0)}\exp(\int_0^t (\sigma(t)-\mu(t))d\widetilde{W}(u) - 0.5\int_0^t (\sigma(t)-\mu(t))^2 du) - \int_0^t \sum_1^d \mu_i(t)(\sigma_i(t)-\mu_i(t))du \tag{11}$$

So we can find a measure

$$d\widetilde{W}^N(u) = \begin{bmatrix} d\widetilde{W}_1 - \mu_1(u) \\ d\widetilde{W}_2 - \mu_2(u) \\ \vdots \\ d\widetilde{W}_d - \mu_d(u) \end{bmatrix} \tag{12}$$

This equation is telling you that under measure N, the normal distribution in risk neutral have positive drift $\mu(u)$

Example 2.1. Under single stock measure, the log normal stock actually have σ^2 drift

$$d(\log(S)) = (r - \frac{1}{2}\sigma^2)dt + \sigma\cdot\sigma dt + \sigma d\widetilde{W}^S(t) \tag{13}$$

Example 2.2. Quanto Option, suppose XAU-EUR is $S_1(t)$ and USD-EUR $S_2(t)$. Then we have

$$\frac{d(S_1)}{S_1} = r_f dt + \sigma_1 d\widetilde{W_1}^f(t) \tag{14}$$

$$\frac{d(S_2)}{S_2} = (r_f - r_d)dt + \sigma_2 d\widetilde{W_2}^f(t) \tag{15}$$

The inverse of exchange rate process under EUR is

$$d(\frac{1}{S_2}) = \frac{1}{S_2}((r_d - r_f + \sigma_2^2)dt - \sigma_2 d\widetilde{W_2}^f(t)) \tag{16}$$

Under EUR risk neutral measure, the XAU by USD have (using $dXY = XdY + Xdx + dXdY$)

$$\frac{dS_3}{S_3} = (r_d + \sigma_2^2 - \rho\sigma_1\sigma_2)dt + \sigma_1 d\widetilde{W_1}^f(t) - \sigma_2 d\widetilde{W_2}^f(t) \tag{17}$$

when we change measure EUR to measure USD then we have

$$\begin{aligned}\frac{dS_3}{S_3} &= (r_d + \sigma_2^2 - \rho\sigma_1\sigma_2)dt + (\sigma_1 - \rho\sigma_2)(d\widetilde{W_1}^d(t) + \rho\sigma_2 dt) \\ &- \sigma_2\sqrt{1-\rho^2}(d\widetilde{W_2}^d(t) + \sqrt{1-\rho^2}\sigma_2 dt \\ &= r_d dt + \sqrt{\sigma_1^2 + \sigma_2^2 - 2\rho\sigma_1\sigma_2}\cdot d\widetilde{W_3}^d(t)\end{aligned} \tag{18}$$

The last equation shows that after exchange rate consideration, the XAU in USD is still a martingale under USD measure.

2.3 Normal Based T Measure

Based on the analysis above. Since discount zero coupon bond is a martingale under risk neutral measure.

$$d(D(t)B(t,T)) = D(t)B(t,T) - \sigma^*(t)d\widetilde{W}(t) \tag{19}$$

Use equation 12 can effciently get

$$d\widetilde{W}^T(t) - \sigma^*(t) = d\widetilde{W}(t) \tag{20}$$

2.4 Use T Measure To Price Option

Even if interest rate is random, we can still get BS formula in a very general way.

$$\widetilde{E}(D(T)(S(T)-K))^+ = \widetilde{E}(D(T)S(T)\mathbb{1}[S(T)>K]) - K\widetilde{E}(D(T)\mathbb{1}[S(T)>K]) \tag{21}$$

The second part is easy to show. The σ here is the vol of forward price, which is comprised of both vol of spot and vol of zero bond(interest rate)

$$\begin{aligned}
K\widetilde{E}(D(T)\mathbb{1}[S(T)>K]) &= KB(0,T)\widetilde{E}^T(F(T,T)>K) \\
&= KB(0,T)P^T(\exp(-\frac{1}{2}\sigma^2 t + \sigma\widetilde{W}^T(T)) > \frac{K}{F(0,T)}) \\
&= KB(0,T)N(d_2)
\end{aligned} \tag{22}$$

The first part is quite similar

$$\widetilde{E}(D(T)S(T)\mathbb{1}[S(T)>K]) = S(0)\widetilde{E}^S(F(T,T)>K) \tag{23}$$

The tricky part is in fact, under the definition of S measure, the inverse of F is a martingale with vol $-\sigma$

$$\frac{B(0,T)}{S(0)} = \frac{1}{F(0,T)} = E^S(\frac{B(t,T)}{S(t)}) = E^S(\frac{1}{F(t,T)}) \tag{24}$$

The using the similar technique as for second part proof we can get the expression for first term.

$$S(0)N(d_1) \tag{25}$$

3 HJM and LMM

3.1 HJM Framework

3.1.1 No Arbitrage Condition

A key things to remember is that all CIR process, Vasicek process are Markov models, and lies in the HJM framework.

$$f(t,T) - f(0,T) = \int_0^t (\alpha(u,T)du + \sigma(u,T))dW(u) \tag{26}$$

At time t, since the zero bond $B(t,T)$ is a martingale.

$$log(B(t,T))) = -\int_t^T f(t,u) \tag{27}$$

$$\begin{aligned} dlog(B(t,T))) &= f(t,t)dt - (\int_t^T \alpha(t,u)du)dt + \int_t^T \sigma(t,u)dW(t) \\ &= (f(t,t) - \alpha^*(t))dt + \sigma^*(t)dW(t) \end{aligned} \tag{28}$$

Then we have

$$\begin{aligned} d(B(t,T))) &= f(t,t)dt - (\int_t^T \alpha(t,u)du)dt + \int_t^T \sigma(t,u)dW(t) \\ &= B(t,T)(f(t,t) - \alpha^*(t) + \frac{1}{2}\sigma^{*2}(t))dt - \sigma^*(t)dW(t) \end{aligned} \tag{29}$$

Then we have

$$\begin{aligned} d(D(t)B(t,T))) &= f(t,t)dt - (\int_t^T \alpha(t,u)du)dt + \int_t^T \sigma(t,u)dW(t) \\ &= D(t)B(t,T)(-\alpha^*(t) + \frac{1}{2}\sigma^{*2}(t))dt - \sigma^*(t)d\widetilde{W}(t) \end{aligned} \tag{30}$$

So we need to have

$$(-\alpha^*(t,T) + \frac{1}{2}\sigma^{*2}(t,T)) = \Theta(t)|forT \in [0,T] \tag{31}$$

Then take derivative with T, we get

$$\alpha(t,T) = \sigma^*(t)\sigma(t,T) \tag{32}$$

3.1.2 Some Important Feature

Once we get no-arbitrage condition, we can have several important conclusion.

First, the T measure under this is vs risk neutral measure.

$$d\widetilde{W}^T(t) - \sigma^*(t) = d\widetilde{W}(t) \tag{33}$$

Second, The forward rate's drift is. This means the forward rate can be solely determined by sigma process.

$$f(t,T) - f(0,T) = \int_0^t (\sigma^*(u)\sigma(u,T)du + \sigma(u,T)dW(u)) \tag{34}$$

3.2 LMM Framework

3.2.1 Forward Payment Replicate

The first things to remember is how to replicate a pay off for a FRA that pays $L(T,T)$ at time $T+\delta$. In fact we can have portfolio of $\frac{1}{\delta}B(t,T) - \frac{1}{\delta}B(t,T+\delta)$ to exactly replicate the payoff.(At time T we need to reinvest our earning of first leg to $B(T,T+\delta)$ and have payoff $\frac{1}{\delta B(T,T+\delta)}$)

In fact, the red part is $B(t,T+\delta)L(t,T)$ and we can think of it as a $T+\delta$ bond times forward rate

3.2.2 $B(t,T+\delta)$ Measure Pricing Example (T measure)

Since $B(t,T+\delta)L(t,T)$ is a price of contract (tradable), so under $B(t,T+\delta)$ measure $L(t,T)$ is a martingale. So for example the price of caplet paying $(L(T,T)-K)^+$ at time $T+\delta$ can be valued as

$$C(t,T) = B(t,T+\delta)E^{T+\delta}(L(T,T)-K)^+ \tag{35}$$

Since at $B(t,T+\delta)L(t,T)$ measure the $L(t,T)$ satisfy $d(L(t,T)) = \gamma(t,T)L(t,T)d\widetilde{W}^{T+\delta}(t)$. Same as HJM, we need to find a vol process for $L(t,T)$ to price the caplet using blacks formula

In fact, we can get the $\gamma(t,T)$ by using zero bond at risk neutral measure. Since $F(t,T) = (\frac{1}{\delta}B(t,T) - \frac{1}{\delta}B(t,T+\delta))/B(t,T+\delta)$

So we have

$$
\begin{aligned}
F(t,T)+\frac{1}{\delta} &= \frac{B(t,T)}{B(t,T+\delta)} \\
&= \frac{B(0,t)}{B(0,T)}\exp\Big(\int_0^t (\sigma^*(t,T)-\sigma^*(t,T+\delta))\cdot d\widetilde{W}(u) \\
&- 0.5\int_0^t \|\sigma^*(t,T)-\sigma^*(t,T+\delta)\|^2\,du \\
&- \int_0^t \sum_1^d \sigma_i^*(t,T+\delta)(\sigma_i^*(t,T)-\sigma_i^*(t,T+\delta)du\Big)
\end{aligned} \tag{36}
$$

So

$$
\begin{aligned}
d(F(t,T)) &= (F(t,T)+\frac{1}{\delta}) \\
&- (\sum_1^d \sigma_i^*(t,T+\delta)(\sigma_i^*(t,T)-\sigma_i^*(t,T+\delta)du+(\sigma^*(t,T)-\sigma^*(t,T+\delta))\cdot d\widetilde{W}(u)) \\
&= (F(t,T)+\frac{1}{\delta})(\sigma^*(t,T)-\sigma^*(t,T+\delta))\cdot d\widetilde{W}^{T+\delta}(u)
\end{aligned} \tag{37}
$$

If we keep HJM notation choose negative σ

$$
\gamma(t,T)=\frac{1+\delta F(t,T)}{F(t,T)}\frac{1}{\delta}(\sigma^*(t,T+\delta)-\sigma^*(t,T)) \tag{38}
$$

This is the vol process under $B(t,T+\delta)$ measure

So you can do PCA on historical vol to get component. Then use implied vol of zero coupon bonds or Caplets to get the factored implied vol. Finally you can use monte-carlo engine to price the exotics in forward rate process. The use that to do the pricing.of Caplets or Swaptions.

3.2.3 Price A Claim Using Terminal Bond Measure

Assume a payoff at time t_n depend on the forward rate at $t1, t2, t3$, and now we are at t. Then we can to use the terminal bond measure (You always have to use a measure to price as base. The example is pay something base on the average past forward rates, which are the use of series of rates for payments)

If we use $B(t,t_n)$ measure to price the forward of the forward contract that pays $F(t_{n-2},t_{n-1})$ at time t_{n-1}. Then we have

$$
E^{B(u,t_n)}\Big(\frac{F(u,t_{n-2},t_{n-1})-F(t,t_{n-2},t_{n-1})}{B(u,t_n)}(1+\delta F(u,t_{n-1},t_n))\Big)|_{u=t_n}=0 \tag{39}
$$

The equation above can be rewrite as

$$
\begin{aligned}
F(t, t_{n-2}, t_{n-1}) &= \frac{E^{B(u,t_n)}_{u=t_n}(F(u, t_{n-2}, t_{n-1})(1 + \delta F(u, t_{n-1}, t_n))}{E^{B(u,t_n)}_{u=t_n}(1 + \delta F(u, t_{n-1}, t_n))} \\
&= \frac{COV^{B(u,t_n)}_{u=t_n}(F(u, t_{n-2}, t_{n-1})(1 + \delta F(u, t_{n-1}, t_n))}{E^{B(u,t_n)}_{u=t_n}(1 + \delta F(u, t_{n-1}, t_n))} + E^{B(u,t_n)}_{u=t_n} F(u, t_{n-2}, t_{n-1}) \\
&= \frac{\delta \rho_{t_{n-2}, t_{n-1}} \sigma_{t_{n-2}} \sigma_{t_{n-1}}}{E^{B(u,t_n)}_{u=t_n}(1 + \delta F(u, t_{n-1}, t_n))} + E^{B(u,t_n)}_{u=t_n} F(u, t_{n-2}, t_{n-1}) \\
&= \frac{\delta \rho_{t_{n-2}, t_{n-1}} \sigma_{t_{n-2}} \sigma_{t_{n-1}}}{1 + \delta F(t, t_{n-1}, t_n))} + E^{B(u,t_n)}_{u=t_n} F(u, t_{n-2}, t_{n-1})
\end{aligned}
\tag{40}
$$

So the drift is $-\frac{\rho_{t_{n-2}, t_{n-1}} \sigma_{t_{n-2}} \sigma_{t_{n-1}}}{1+\delta F(t, t_{n-1}, t_n))}$

4 Real Data Calibration

4.1 LMM

To calibrate LMM, we need two kinds of data: volatility and correlation of forward rate.

- Caps are for calculating volatility (sum of individual vol)
- Swaptions are for calulating volatility and correlation (average of forward rate)
- Correlation can also be derived from historical analysis

5 Numerical Method

5.1 Tree Method

5.1.1 Binary Tree Method

To price the American option, we can use tree method (Binomial/Trinomial). Suppose the we have σ, r, δt, P_+, P_-. The we have $u = e^{\sigma\sqrt{\delta t}}$, $d = e^{-\sigma\sqrt{\delta t}}$ as multiplication factors. We can have

$$\begin{aligned} P^+ &= \frac{e^{r\delta t} - d}{u - d} \\ P^- &= \frac{u - e^{r\delta t}}{u - d} \end{aligned} \tag{41}$$

To summarize the method

1. Time complexity $O(n^2)$
2. Good risk neutral interpretation
3. Need to choose small enough δt to make the P positive (not flexible enough)
4. the range of dS is determined by δt (hard to select δt)

5.1.2 Trinomial Tree Method

5.2 Acceptance Rejection

For a distribution which only has pdf but no cdf, we can get a distribution with cdf and pdf $g(v)$ generated by $F^{-1}(u)$ with u from $[0, 1]$. (We need to know F(x) is uniform distribution for x belongs to any distribution. Since quantile by definition is uniform distributed)

We get a big number M to accept y generated by $g(y)$ and v from uniform if

$$v <= \frac{f(y)}{Mg(y)}$$

This can be thought of as we generate density g(y) and only accept those density weighted by possibility $f(y)$.

5.3 PDE numerical solver

6 Coding Basics

6.1 C++

6.1.1 Template

Key things to know for template:

1. as long as the type is specified clearly, template class will be initiated at compile type
2. you can write a factorial calculation at using template at complied time. also constexpr will achieve this function as well
3. we can achieve virtual function by using template

```
#include<iostream>
#include<string>
using namespace std;

template <typename T> void show(const T& data){
  if(typeid(data) == typeid(string("abc")))
    cout << "This is a string" << endl;
  if(typeid(data) == typeid(1))
    cout << "This is a int" << endl;
}

int main(){
  show(8);
  show(string("abc"));
}
```

6.1.2 Variable Dict

Use variable dict template to write print function

```
#include<iostream>
#include<string>
using namespace std;

template<typename T> void print(const T&t){
  cout << t << endl;
}

template<typename T, typename... Y> void print(const T& first, Y... y){
  cout << first << " ";
  print(y...);
}

int main(){
  print(1, 2, 3, "abc");
}
```

6.1.3 Perfect Forwarding

6.1.4 Smart Pointer

6.2 Python

6.2.1 Dataframe Data-structure

6.3 Java

6.4 UNIX Bash

6.5 MongoDB

7 Coding Design

7.1 SOLID Principles

Solid principles are the most basic principles in system design

7.1.1 Single Responsible principle

Suppose you are going to design two classes, *ractangle* and *triangle*.

```
class shape{
private:
        int width;
        int height;
};

class rectangle: shape {

};

class triangle: shape{
};
```

Then you want to have a factory function to calculate the size. The function itself should only return the size. But you are not satisfied with it, you want to return an object with other functions.

```
struct summary{
        int size;
        string format;
        string to_string(){
                if(format == ".2d")
                        return ....
        }
};

struct getSize(const shape* s, const string& format){
}
```

However, this is not a good design since it violate the single responsibility principle. Since now getSize is not only calculate the size, but also in charge of formatting the output. Which add extra interconnection and logic to the code. So we need to refactor the code into following

```
struct summary{
        int size;
        string format;
        string to_string(){
```

```
                if(format == ".2d")
                        return ....
        }
};

summary getFormatObject(int size, const string& format){
}

struct getSize(const shape* s){
}
```

The single responsible principle tell us to write a function only for a single purpose. Do not control to many things.

7.1.2 Open/Close principle

Inside the get size function, a short cut. We can write the function with bunch of if-else:

```
struct getSize(const shape* s){

        // dynamic_cast here can check the actual type safely
        // at the run time

        if ( dynamic_cast<rectangle*>( s ) )
        // do something

        else if ( dynamic_cast<triangle*>( s ) )
        // do something else
        }
}
```

The function above is not a good way of code because it violate the open/close principle. If you have another class circle, then you will have to **modify** the function method. **open** in the principle means open for extension (add more method) while **close** here means close for modification of the old method.

```
class circle: shape {

};

int getSize(const shape* s){

        if ( dynamic_cast<rectangle*>( s ) ){
        }
        else if ( dynamic_cast<triangle*>( s ) )
        }
        else if ( dynamic_cast<circle*>( s ) )
```

```
        // modify the getSize every time you add new class features
        }
}
```

Correct way of coding is add getSize method as a virtual member function of base and derived class

```
class shape{
private:
        int width;
        int height;
public:
        virtual int getSize() const;
};

class rectangle: shape {
        int getSize() const{
        }
};

class triangle: shape {
        int getSize() const{
        }
};

class circle: shape {
        int getSize() const{
        }
};

int getSize(const shape* s){

        return s->getSize();
}
```

Then every method is closed. If we want to have new class, no existing code subsection will be modified.

7.1.3 Liskov Substitution principle

Liskov substitution principle is about how to design base and child to be inter-changeable. The famous eclipse-circle problem is

```
class eclipse{
private:
        float a;
        float b;
```

```
public:
        eclipse(float a, float b): a(a), b(b){}
}

class circle: eclipse{
public:
        circle(float r): eclipse(r, r){}
private:
}
```

Some people would say it is a good design. But, to Liskov principle tell us that the child class must provide everything that parent class method. Think about if our eclipse class have a method **stretch**

```
class eclipse{
private:
        float a;
        float b;
public:
        eclipse(float a, float b): a(a), b(b){}
        stretch(float a, float b){this-> a = a; this->b = b;}
}
```

Then our circle must have this method too. However, circle cannot have this method. So here are two possible solutions:

- throw exception or return failure value in circle stretch method
- redefine the data model, add another common base class "shape" to both of it. Now circle vs eclipse is D vs D (Derived vs Derived)

7.1.4 Interface Segregation Principle

In a short word, base interface should not have too much functions, which means the derived class should never be forced to have the method it does not use.

```
class shape{
private:
        int width;
        int height;
public:
        virtual int getSize() const;
        virtual int getVolume
};

class rectangle: shape {
/*
        rectangle here is forced to implement a getVolume.
```

```
*/
        int getSize() const{
        }
};


class cubic: shape {
        int getVolume const{
        }
};
```

A good solution is bifurcate the original shape interface into two parallel sub interface class. Then we segregate methods that have conflict.

```
class BaseShape{
private:
        int width;
        int height;
};

class FlatShape: BaseShape{
/*
        segregate the getVolume and getSize into two branch
*/
        virtual int getSize const(){
        }
}

class SolidShape: BaseShape{
        virtual int getVolume const(){
        }
}

class rectangle: FlatShape {
        int getSize() const{
        }
};

class cubic: SolidShape {
        int getVolume const{
        }
};
```

7.1.5 Dependency Inversion Principle

This is the easiest of five. The higher level function should only rely on the lower level abstractions but not the detailed implementation. The bridge pattern, abstract factory pattern are good example. Here the high-level user paint depend on the abstraction of different Painter, no matter how painter is changed. The rectangle can always get the updated result.

Adding abstraction dependency is called dependency injection.

```
class Painter:{
public:
        virtual void paint(){}
};

class FancyPainter: Painter{
public:
        void paint(){
        // fancy painting
        }
};

class rectangle: shape{

// high level user rectangle only need to know we have a member
// which is in charge of paint. That's it, no more info that high-level
// user need to know.

public:
        rectangle(float x, float y, Painter * pt): width(x), width(y), pt(pt){
                // add Painter* here is actually called dependency injection
        }

        void paint(){
                pt.paint();
        }
private:
        Painter pt;
};
```