Codes We used:

Since we combine our code with example code of lecture 4, we decide to give prof you a clear exhibition of our code

First Part: Construct Implied Tree

#include "FunctionNeed.h"

```
//
//  FunctionNeed.h
//  TEst
//
//  Created by 陈家豪 on 2020/4/13.
//  Copyright © 2020 陈家豪. All rights reserved.
//

#ifndef FunctionNeed_h
#define FunctionNeed_h
#include <cmath>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/io.hpp>
#include "Spline.h"
#include "CRR.h"
namespace ublas = boost::numeric::ublas;
using namespace std;



//This class is made for store implied volatility data

class ImpVol
{
public:
    ublas::vector<double> time;
    ublas::vector<double> strike;
    ublas::vector<double> ivol;
public:
    ImpVol(): time(3000),strike(3000),ivol(3000) {}

public:
    ublas::vector<double> GetStrike() const
```

```
    {
        ublas::vector<double> newvect(50);
        for(int i = 0;i<50;i++)
        {
            newvect[i] = strike[i];
        }
        return newvect;
    }

};




// This is only used in debug.
void ShowMatrix(ublas::matrix<double> mat)
{
    std::cout<<"----------------Show Matrix-----------------"
<<std::endl;
    for(int i = 0;i<mat.size1();i++)
    {
        for(int j=0;j<mat.size2();j++)
        {
            std::cout << mat(i,j) << "  , ";
        }
        std::cout <<std::endl;
    }
    std::cout<<"--------------------------------------------"
<<std::endl;
}




// search function to find the interval of T (maturity)
int search(ublas::vector<double> time,double mat)
{
    int index = 0;
    for (int i = 0; i<=3000; i = i+50) //3000 can be time.size()
    {

        if(mat>=time[i])
        {
            index = i;
        }

        else
        {
```

```
            break;
        }
    }
    return index;
}
```

```
//vector slice
ublas::vector<double> slice(ublas::vector<double> vect, int index,int
step)
{
    ublas::vector<double> newvect(step);
    int k = 0;
    int end = index+step;
    for (int i=index; i<end; i++)
    {
        newvect[k] = vect[i];
        k++;
    }
    return newvect;
}
```

```
//Linear interpolation
ublas::vector<double> interpol(double T, ublas::vector<double>
var1,double T1,ublas::vector<double> var2, double T2)
{
    return var1 + (T-T1)*(var2-var1)/(T2-T1);
}
```

```
//sqrt to vector
ublas::vector<double> sqrt2vec(ublas::vector<double> vec)
{
    ublas::vector<double> newvec(vec.size());
    for (int i=0; i < vec.size(); i++)
    {
        newvec[i] = sqrtf(vec[i]);
    }
    return newvec;
}
```

```
//change ublas vector to std vector for spline function
vector<double> ublas2std(ublas::vector<double> vec)
{
    vector<double> newvec(vec.size());
    for (int i=0; i < vec.size(); i++)
    {
        newvec[i] = vec[i];
    }
    return newvec;
}


//Given impvol, T and strike, calculate implied volatility
double GetImpVol(const ImpVol& impvol,double T, double
K,ublas::vector<double> strike)
{
    double ind_1; double ind_2;
    int ind = search(impvol.time, T);
    if(ind < 2950 )
    {
        ind_1 = ind;
        ind_2 = ind+50;
    }
    else
    {
        ind_1 = ind -50;
        ind_2 = ind;
    }

    ublas::vector<double> sig1 = slice(impvol.ivol, ind_1, 50);
    ublas::vector<double> var1 = ublas::element_prod(sig1, sig1);
    //std::cout << "var 1: "<< var1 << std::endl;

    ublas::vector<double> sig2 = slice(impvol.ivol, ind_2, 50);
    ublas::vector<double> var2 = ublas::element_prod(sig2, sig2);
    //std::cout << "var 2: "<< var1 << std::endl;

    double T1 = impvol.time[ind_1];
    double T2 = impvol.time[ind_2];
    ublas::vector<double> var = interpol(T, var1, T1, var2, T2);
    //std::cout << "var: "<<var << std::endl;

    ublas::vector<double> sig = sqrt2vec(var);
    //std::cout << "sig: "<<sig << std::endl;

    tk::spline s;
    vector<double> std_strike = ublas2std(strike);
```

```
    vector<double> std_sig = ublas2std(sig);
    s.set_points(std_strike,std_sig);
    //printf("volatility spline at %f is %f\n", K, s(K));

    return s(K);
}



//Made for modify wrong stock price
double UpperModification(ublas::matrix<double> stock,int k,int
i,double r,double q,double dt)
{
    if(i>0)
    {
        if (stock(k,i)>stock(k-1,i) and stock(k,i) < stock(k-1,i-1)
and stock(k,i)>0)
        {
            //std::cout<<"Modification Stock: " << stock << std::endl;
            return stock(k,i);
        }
        else
        {
            stock(k,i) = stock(k-1,i)*stock(k,i+1)/stock(k-1,i+1);
            if(stock(k,i)>stock(k-1,i) and stock(k,i) < stock(k-1,i-1)
and stock(k,i)>0){
                //std::cout<<"Modification Stock: " << stock <<
std::endl;
                return stock(k,i);
            }
            else{
                double df_ = std::exp((r-q)*dt);
                double fw_1 = stock(k-1,i-1)*df_;
                double fw_2 = stock(k-1,i)*df_;
                stock(k,i) = (fw_1+fw_2)/2;
                //std::cout<<"Modification Stock: " << stock <<
std::endl;
                return stock(k,i);
            }
        }
    }
    else
    {
        if (stock(k,i)>stock(k,i+1) and stock(k,i) > stock(k-1,i) and
stock(k,i)>0) {
            //std::cout<<"Modification Stock: " << stock << std::endl;
            return stock(k,i);
        }
        else{
```

```
            stock(k,i) = stock(k-1,i)*stock(k,i+1)/stock(k-1,i+1);
            //std::cout<<"Modification Stock: " << stock << std::endl;
            return stock(k,i);
        }
    }
}


//Made for modify wrong stock price
double DownModification(ublas::matrix<double> stock,int k,int i,double
r,double q,double dt)
{
    if(i<k)
    {
        if (stock(k,i)>stock(k-1,i) and stock(k,i) < stock(k-1,i-1)
and stock(k,i)>0)
        {
            //std::cout<<"Modification Stock: " << stock << std::endl;
            return stock(k,i);
        }
        else
        {

            stock(k,i) = stock(k-1,i-1)*stock(k,i-1)/stock(k-1,i-2);
            if(stock(k,i)>stock(k-1,i) and stock(k,i) < stock(k-1,i-
1)){
                //std::cout<<"Modification Stock: " << stock <<
std::endl;
                return stock(k,i);
            }
            else{
                double df_ = std::exp((r-q)*dt);
                double fw_1 = stock(k-1,i-1)*df_;
                double fw_2 = stock(k-1,i)*df_;
                stock(k,i) = (fw_1+fw_2)/2;
                //std::cout<<"Modification Stock: " << stock <<
std::endl;
                return stock(k,i);
            }
        }
    }
    else
    {
        if (stock(k,i)<stock(k,i-1) and stock(k,i) < stock(k-1,i-1)
and stock(k,i)>0)
        {
            //std::cout<<"Modification Stock: " << stock << std::endl;
            return stock(k,i);
```

```
        }
        else{
            stock(k,i) = stock(k-1,i-1)*stock(k,i-1)/stock(k-1,i-2);
            //std::cout<<"Modification Stock: " << stock << std::endl;
            return stock(k,i);
        }
    }
}



//Made for modify wrong stock price
double OddModification(ublas::matrix<double> stock,int k,int i,double
r,double q,double dt)
{
    if (stock(k,i)>stock(k-1,i) && stock(k,i) < stock(k-1,i-1) &&
stock(k,i)>0)
    {
        return stock(k,i);
    }
    else
    {
        double df_ = std::exp((r-q)*dt);
        double fw_1 = stock(k-1,i-1)*df_;
        double fw_2 = stock(k-1,i)*df_;
        stock(k,i) = (fw_1+fw_2)/2;
        return stock(k,i);
    }
}




//In order to return our matrix
struct Result {
    ublas::matrix<double> stock;
    ublas::matrix<double> p;
    ublas::matrix<double> forward;
    ublas::matrix<double> lambda;
};



//Main part of tree generating process
struct Result ImpliedTree(const ImpVol& impvol,double T,double
s0,double r,double q,int N)
{
```

```
    Result set;
    double dt = T/N;
    double df_ = std::exp(r*dt);
    double df = std::exp((r-q)*dt);
    ublas::matrix<double> p(N+1,N+1);
    ublas::matrix<double> lambda(N+1,N+1);
    ublas::matrix<double> stock(N+1,N+1);
    ublas::matrix<double> forward(N+1,N+1);
    stock(0,0) = s0;
    lambda(0,0) = 1;
    p(0,0) = 0;
    forward(0,0) = s0*df;



//    get strike
    ublas::vector<double> strike = impvol.GetStrike();
    //std::cout << "strike: " << strike<<std::endl;
    //std::cout << "xxxxxxxxxxxxxxxxBasic Setting
Donexxxxxxxxxxxxxxxxx " << std::endl;
//          ------------------------------------------------------------
--
//    forward induction
    for (int k = 1; k <= N; k++)
    {
        int ind_1;
        int ind_2;
        int ind_f = floor(k/2);

//    Odd or Even
        if (k % 2 == 0){
            stock(k,k/2) = s0;
            //std::cout << "stock_1 even situation: "  <<std::endl;
            //ShowMatrix(stock);
            ind_1 = k/2-1;
            ind_2 = k/2;
        }
        else{
            ind_1 = floor(k/2)-1;
            ind_2 = floor(k/2)+1;
            // Get Implied Volatility
            double iv = GetImpVol(impvol, k*dt, s0, strike);

            // Get Call Price
            //EuropeanOption euroCall(Call, s0, Date(2016, 1, 1));
            //double Call_Price = CRR(C, s0, k*dt, s0, iv, r,q, k);
            double Call_Price = bsformula(C, s0, k * dt, s0, iv, r,
q);
            //std::cout << "Call_Price: " << Call_Price << std::endl;
```

```
            // Get SUM part
            double sum = 0.0;

            for(int j = 0;j < ind_f;j++)
            {
                sum =sum + lambda(k-1,j)*(forward(k-1,j)-stock(k-
1,ind_1));
            }

            // Some parts
            double a = df_*Call_Price;

            double b = lambda(k-1,ind_f)*s0-sum;
            //std::cout << "Lambda " << lambda(k-1,ind_f)<< std::endl;
            double numerator = s0*(a+b);
            //std::cout << "Numerator: " << numerator << std::endl;
            double c = lambda(k-1,ind_f)*s0*df_;
            //std::cout << "part c: " << c << std::endl;
            double denominator = c - a + sum;
            //std::cout << "s[k][ind_f]: " << numerator/denominator <<
std::endl;
            if(k<=1)
            {
                stock(k,ind_f) = numerator/denominator;
                stock(k,ind_f+1) = s0*s0/stock(k,ind_f);
            }
            else{
                stock(k,ind_f) = numerator/denominator;
                stock(k,ind_f) = OddModification(stock, k, ind_f, r,
q, dt);
                //ShowMatrix(stock);

                stock(k,ind_f+1) = s0*s0/stock(k,ind_f);
                stock(k,ind_f+1) = OddModification(stock, k, ind_f+1,
r, q, dt);
                //ShowMatrix(stock);
            }
            //std::cout << "stock_2 odd situation: "  << std::endl;
            //ShowMatrix(stock);
        }
        //std::cout << "xxxxxxxxxxxxxxxxxxxxxxxxxxxx Looooooooooop in
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx " << std::endl;

  //          ------------------------------------------------------
----
            // Iteration from formula(13)

            for (int i = ind_1; i >= 0; i--)
            {
```

```
                double stock_ = stock(k-1,i);
                double Maturity = k*dt;

                // Get Implied Volatility
                double iv = GetImpVol(impvol, Maturity, stock_,
strike);

                // Get Call Price
                //double Call_Price = CRR(C, stock_, Maturity, s0, iv,
r,q, k);
                double Call_Price = bsformula(C, stock_, Maturity, s0,
iv, r, q);
                // Get Forward F(k-1,i)
                forward(k-1,i) = stock_*df;

                // Get SUM part
                double sum = 0.0;
                for(int j = 0;j < i;j++)
                {
                    sum =sum + lambda(k-1,j)*(forward(k-1,j)-stock(k-
1,i));
                }

                // Some parts
                double a = df_*Call_Price - sum;
                double b = lambda(k-1,i)*(forward(k-1,i)-
stock(k,i+1));



                //std::cout << "Matrix stock " << std::endl;
                stock(k,i) = (stock(k,i+1)*a-stock(k-1,i)*b)/(a-b);
//get stock
                //std::cout<< "Before Modification: " << std::endl;
                //ShowMatrix(stock);
                stock(k,i) = UpperModification(stock, k, i, r, q, dt);
                //std::cout << "After Modification: " << std::endl;
                //ShowMatrix(stock);


            }
        //std::cout << "--------------------------------Formula 13
out--------------------------------" << std::endl;
```

```
      //          ------------------------------------------------
--------
            // Iteration from formula(14)
            for (int i = ind_2; i < k; i++)
            {
                double stock_ = stock(k-1,i); //stock_ mean strike
need interpolation
                double Maturity = k*dt;

                // Get Implied Volatility
                double iv = GetImpVol(impvol, Maturity, stock_,
strike);

                // Get Put Price
                //double Put_Price = CRR(P, stock_, Maturity, s0, iv,
r,q, k);
                double Put_Price = bsformula(P, stock_, Maturity, s0,
iv, r, q);
                // Get Forward F(k-1,i)
                forward(k-1,i) = stock_*df;

                // Get SUM part
                double sum = 0.0;
                for(int j = i+1;j < k; j++)
                {
                    sum =sum + lambda(k-1,j)*(stock(k-1,i)-forward(k-
1,j));
                }

                // Some parts for calculating stock(k,i+1)
                double a = df_*Put_Price - sum;
                double b = lambda(k-1,i)*(forward(k-1,i)-stock(k,i));


                //std::cout << "Matrix stock: " << std::endl;
                stock(k,i+1) = (stock(k,i)*a+stock(k-
1,i)*b)/(a+b);//get martix
                //std::cout<< "Before Modification: " << std::endl;
                //ShowMatrix(stock);
                stock(k,i+1) = DownModification(stock, k, i+1, r, q,
dt);
                //std::cout << "After Modification: " << std::endl;
                //ShowMatrix(stock);

            }
       //std::cout << "-------------------------Formula 14 out-----
-----------------------" << std::endl;
```

```
      //          ------------------------------------------------
--------
        // Get probability
        forward = stock*df;
        for (int i=0; i<k; i++)
        {
            p(k,i) = (forward(k-1,i) - stock(k,i+1))/(stock(k,i)-
stock(k,i+1));
            if (p(k, i) < 0 || p(k, i) > 1) {
                p(k, i) = 0.5;
            }
            //std::cout << "probability: " << p << std::endl;
        }

        // Get Lambda
        for(int i = 0;i<=k;i++)
            {
                if (i == 0) {
                    lambda(k,0) = std::exp(-r*dt)*(p(k,0)*lambda(k-
1,0));
                }
                else if (i == k){
                    lambda(k,k) = std::exp(-r*dt)*lambda(k-1,k-1)*(1-
p(k,k-1));
                }
                else
                {
                    lambda(k,i) = std::exp(-r*dt)*((1-p(k,i-
1))*lambda(k-1,i-1)+p(k,i)*lambda(k-1,i));
                }
                //std::cout << "lambda: " << lambda << std::endl;
            }
    }


    //std::cout << "--------------------Loop finished---------------
-------" << std::endl;

  set.stock = stock;
    set.lambda = lambda;
    set.p = p;
    set.forward = forward;

    //std::cout << "stock price: " << stock << std::endl;
    //std::cout << "lambda: " << lambda << std::endl;
    //std::cout << "p: " << p << std::endl;
    //std::cout << "forward: " << forward << std::endl;
```

```
    return set;
}
#endif /* FunctionNeed_h */
```

Second Part: Implied Tree Pricer

```
//TreePricer.h
double impliedTree_pricer(const Market& market, const TreeProduct&
trade, int N, const ImpVol& impvol,double q = 0)
{

    // set up the tree model and parameters
    double T = trade.GetExpiry() - market.asof;
    double rate = market.interestRate;
    double S0 = market.stockPrice;
    std::vector<double> states(N + 1);
    double dt = T / N;
    // initialize the final states, apply payoff directly
    Result set = ImpliedTree(impvol, T, S0, rate, q, N);
    for (int i = 0; i <= N; i++) {
        states[i] = trade.Payoff(set.stock(N, i));
    }
    for (int k = N - 1; k >= 0; k--)
        for (int i = 0; i <= k; i++) {
            // calculate continuation value
            double df = exp(-rate * dt);
            double p = set.p(k+1, i);
            //cout << "p:" << p << "at" << k << "at" << i << endl;
            double continuation = df * (states[i] * p + states[i + 1]
* (1 - p));
            //cout << "zhe:" << states[i] * p + states[i + 1] * (1 -
p) << endl;
            //cout << "rate" << df;
            //cout << "value:" << continuation << endl;
            // calculate the option value at node(k, i)
            double S = set.stock(k, i);
            states[i] = trade.ValueAtNode(S, dt * k, continuation);
            //cout << "node:" << states[i] << endl;
        }
    return states[0];
}
```

Third Part: Implementation Implied Tree Pricer

```
//main.cpp
void test_impliedTreePricer()
{

    std::ifstream din;
    din.open("impliedvol.txt");
    if (!din.is_open()) {
        std::cout << "no";
    }
    std::string line;
    ImpVol impvol;

    int i = 0;

    while (getline(din, line))
    {

        char a = '\t';
        std::vector<std::string> fields;
        fields = split(line, a);
        impvol.time[i] = stod(fields[0]);
        impvol.strike[i] = stod(fields[1]);
        impvol.ivol[i] = stod(fields[2]);
        i = i + 1;
    }//  ----------------------Load Implied Volatility-------------
-----------


    std::ifstream fin("simpleMkt.dat");
    if (fin) {
        Market mkt;
        fin >> mkt;
        mkt.volatility = 0.1444;
        mkt.stockPrice = 1.25;
        mkt.interestRate = 0.01;
        std::vector<double> Nrange(10);
        std::vector<double> CRR_price(10);
        std::vector<double> Imp_price(10);
        cout << "const volatility:\t" << mkt.volatility << endl;
        cout << "stock price:\t\t" << mkt.volatility << endl;
        cout << "interest rate:\t\t" << mkt.volatility << endl;
        int nTenors = 10;


        std::cout << "\n------------ European Option ---------------
-------\n";
        for (int i = 0; i < nTenors; i++) {
            Nrange[i] = 10 + i * 20;
            EuropeanOption euroCall(Call, 1.2, Date(2016, 1, 1));
```

```
            Imp_price[i] = impliedTree_pricer(mkt, euroCall,
Nrange[i], impvol);
            CRRBinomialTreePricer crrBTreePricer(Nrange[i]);
            //std::cout << crrBTreePricer.Price(mkt, euroCall) <<
std::endl;
            CRR_price[i] = crrBTreePricer.Price(mkt, euroCall);
            //std::cout << bsformula( C,1.2, 1, 1.25, 0.13, 0.01, 0)
<< std::endl;
        }
        ShowArray2(Nrange,CRR_price, Imp_price, nTenors);


        std::cout << "\n---------------- American Option ------------
------\n";
        for (int i = 0; i < nTenors; i++) {
            Nrange[i] = 10 + i * 20;
            AmericanOption amCall(Call, 1.2, Date(2016, 1, 1));
            Imp_price[i] = impliedTree_pricer(mkt, amCall, Nrange[i],
impvol);
            CRRBinomialTreePricer crrBTreePricer(Nrange[i]);
            //std::cout << crrBTreePricer.Price(mkt, euroCall) <<
std::endl;
            CRR_price[i] = crrBTreePricer.Price(mkt, amCall);
            //std::cout << bsformula( C,1.2, 1, 1.25, 0.13, 0.01, 0)
<< std::endl;
        }
        ShowArray2(Nrange, CRR_price, Imp_price, nTenors);


        std::cout << "\n------------------ Barrier Option ------------
---------\n";
        for (int i = 0; i < nTenors; i++) {
            Nrange[i] = 10 + i * 20;
            EuropeanOption euroCall(Call, 1.2, Date(2016, 1, 1));
            BarrierOption barOption(DownBarrier(1.2), euroCall);
            Imp_price[i] = impliedTree_pricer(mkt, barOption,
Nrange[i], impvol);
            CRRBinomialTreePricer crrBTreePricer(Nrange[i]);
            //std::cout << crrBTreePricer.Price(mkt, euroCall) <<
std::endl;
            CRR_price[i] = crrBTreePricer.Price(mkt, barOption);
            //std::cout << bsformula( C,1.2, 1, 1.25, 0.13, 0.01, 0)
<< std::endl;
         }
        ShowArray2(Nrange, CRR_price, Imp_price, nTenors);
    }
}
```