//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
//Project Name: Equity Option Pricing and Risk Management  
//Version: 1.0  
//Project Description: Calculates Equity Option Prices, Option Greeks - Delta, Vega, Gamma, Theta, Rho, does Vol Smile Fitting based on  
//        user inputted Volatility/ Strike grid  
//Date: 21-Aug-2019  
//Owner: Vivek Kumar Rauniyar  
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
// Including C++ libraries  
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#define \_USE\_MATH\_DEFINES  
#include <iostream>  
#include <cmath>  
#include <iomanip>  
#include <bits/stdc++.h>

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
// An approximation to the cumulative distribution function for the standard normal distribution  
//This is a recursive function. I have used this piece of code from internet  
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

double norm\_cdf(const double& x) {  
    double k = 1.0/(1.0 + 0.2316419\*x);  
    double k\_sum = k\*(0.319381530 + k\*(-0.356563782 + k\*(1.781477937 + k\*(-1.821255978 + 1.330274429\*k))));

    if (x >= 0.0) {  
        return (1.0 - (1.0/(pow(2\*M\_PI,0.5)))\*exp(-0.5\*x\*x) \* k\_sum);  
    } else {  
        return 1.0 - norm\_cdf(-x);  
    }  
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
// This calculates d\_j, for j in {1,2}. This term appears in the closed form solution for the European call or put price  
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

double d\_j(const int& j, const double& S, const double& K, const double& r, const double& v, const double& t) {  
    return (log(S/K) + (r + (pow(-1,j-1))\*0.5\*v\*v)\*t)/(v\*(pow(t,0.5)));  
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
//This calculates Call option prices based on user input  
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

double Call\_Opt\_Pricer (const double& S,const double& K,const double& r,const double& v,const double& t)  
{  
 double Call\_Opt\_Premium;  
   
 Call\_Opt\_Premium = S\*norm\_cdf(d\_j(1,S,K,r,v,t)) - K\*exp(-r\*t)\*norm\_cdf(d\_j(2,S,K,r,v,t));  
   
 return Call\_Opt\_Premium;  
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
//This calculates Put option prices based on user input  
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

double Put\_Opt\_Pricer (const double& S,const double& K,const double& r,const double& v,const double& t)  
{  
 double Put\_Opt\_Premium;  
   
 Put\_Opt\_Premium = K\*exp(-r\*t)\*norm\_cdf(-d\_j(2,S,K,r,v,t)) - S\*norm\_cdf(-d\_j(1,S,K,r,v,t));  
   
 return Put\_Opt\_Premium;  
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
//This calculates option price based on tweaked inputs for calculating Option Greeks for the Option whose prices are   
//calculated in the former Option Pricing function calls  
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

double Greeks (const double& S,const double& K,const double& r,const double& v,const double& t, const char& Opt\_type)  
{  
 double New\_Opt\_Premium;  
   
 if(Opt\_type=='P')  
   {  
    New\_Opt\_Premium = Put\_Opt\_Pricer(S,K,r,v,t) ;  
      
   }else if(Opt\_type=='C'){   
      
    New\_Opt\_Premium = Call\_Opt\_Pricer(S,K,r,v,t);  
      
   }   
 return New\_Opt\_Premium;     
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
//This aims at arriving best fit for Volatility Surface based on user inputted Volatility/Strike grid and returns a pointer to the  
//array of calibrated parameters  
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

double \* Vol\_Smile\_Fitting ()  
{

   double Vol[20][20],SABR\_param[10];  
   double ATM\_Vol,Volatility;  
   int \* loc;  
   double ATM\_Strike,Strike;  
   double Strike\_Increment,Strike\_Range,Strike\_Range\_Low,Strike\_Range\_High,Strike\_count;  
   double A,B,C;  
   double A\_high=10,B\_high=10,C\_high=10;  
   double A\_low=-10.0,B\_low=-10.0,C\_low=-10.0;  
   double A\_Fitted,B\_Fitted,C\_Fitted;  
   int ATM;  
     
           
   std::cout << "Enter ATM Strike\n";  
   std::cin >> ATM\_Strike;  
     
   std::cout << "Enter Strike Increment for the Smile Grid. Please enter multiples of 5.\n";  
   std::cin >> Strike\_Increment;  
     
   std::cout << "Enter Lowest Strike for the Strike Range for the Smile Grid. Please enter multiples of 5.\n";  
   std::cin >> Strike\_Range\_Low;  
     
   std::cout << "Enter Highest Strike for the Strike Range for the Smile Grid. Please enter multiples of 5.\n";  
   std::cin >> Strike\_Range\_High;  
     
     
   Strike\_count = (Strike\_Range\_High - Strike\_Range\_Low)/Strike\_Increment+1;  
     
   Strike = Strike\_Range\_Low;  
     
   std::cout << Strike\_count << "      " << Strike << std::endl;  
     
   for(int i=0;i<Strike\_count;i++){  
     
    Vol[0][i]=Strike/ATM\_Strike;    
    std::cout << "\n";  
      
    std::cout << "Enter Implied Vol in % for the Strike  : " << Strike << std::endl;  
    std::cin >> Volatility;  
    Vol[1][i]=Volatility/100;  
      
      
     if(Strike==ATM\_Strike){  
      ATM=i;  
      ATM\_Vol=Volatility/100;  
      }  
         
    Strike = Strike + Strike\_Increment;  
      
   }  
     
     
     
 /////////////////////////////////////////////////////////////////////////////////////////////////    
 //  Vol Smile Fitting using quadratic equation    
 ////////////////////////////////////////////////////////////////////////////////////////////////  
     
   double Min\_Std\_err=1, Std\_err=0,Err=0, ATM\_Diff=0.2,ATM\_Lowest\_Diff=0.5,A\_ATM\_lowest=0,B\_ATM\_lowest=0,C\_ATM\_lowest=0;  
   double Min\_Std\_err\_ATM\_Fixed=1,A\_Fitted\_ATM\_Fixed,B\_Fitted\_ATM\_Fixed,C\_Fitted\_ATM\_Fixed;  
     
   for(A=A\_low;A<=A\_high;A+=0.05){  
    for(B=B\_low;B<=B\_high;B+=0.05){  
     for(C=C\_low;C<=C\_high;C+=0.05){  
        
      Std\_err=0;  
      Err=0;  
        
      for(int i=0;i<Strike\_count;i++){  
           
         if(Vol[0][i]!=1){  
         Err=pow((Vol[1][i]-(A\*pow(Vol[0][i],2.0)+B\*Vol[0][i]+C)),2.0)\*(1/sqrt(abs(ATM\_Strike\*Vol[0][i]-ATM\_Strike)));  
         }  
         if(Vol[0][i]==1){  
         Err=pow((Vol[1][i]-(A\*pow(Vol[0][i],2.0)+B\*Vol[0][i]+C)),2.0);  
         }  
         Std\_err=Std\_err+Err;   
      }  
        
        
      Std\_err= pow(Std\_err,0.5)/Strike\_count;  
        
              
      for(int i=0;i<Strike\_count;i++){  
       Vol[3][i]=A\*pow(Vol[0][i],2.0)+B\*Vol[0][i]+C;  
       }  
        
      if(Min\_Std\_err>Std\_err){  
       Min\_Std\_err=Std\_err;  
       A\_Fitted = A;  
       B\_Fitted = B;  
       C\_Fitted = C;  
      }  
     }  
    }  
   }    
    
   for(int i=0;i<Strike\_count;i++){  
    Vol[2][i]=A\_Fitted\*pow(Vol[0][i],2.0)+B\_Fitted\*Vol[0][i]+C\_Fitted;    
   }

   SABR\_param[0]=A\_Fitted;  
   SABR\_param[1]=B\_Fitted;  
   SABR\_param[2]=C\_Fitted;  
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 
     
   std::cout << "\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  
   std::cout << "The best fit estimates for the parameters are the below\n";  
   std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  
   std::cout << "A   :" << A\_Fitted << std::endl;  
   std::cout << "B   :" << B\_Fitted << std::endl;  
   std::cout << "C   :" << C\_Fitted << std::endl;  
   std::cout << "Minimum Standard error   :" << Min\_Std\_err << std::endl;  
   
   std::cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  
   std::cout << "\nRelative Strike        Implied Vol           Fitted\_Vol            " << std::endl;  
   std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  
     
   for (int i=0;i<Strike\_count;i++){  
      std::cout << "     "<< std::setprecision(6) << Vol[0][i] << "                "<< std::setprecision(6)<< Vol[1][i] << "                "<< std::setprecision(6) << Vol[2][i] << std::endl;  
   }  
     
     
      
   return SABR\_param;  
     
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
//MAIN Function: Takes in user inputs for Option Prices, calculates Option Greeks and prints various calculated outputs  
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(int argc, char \*\*argv)  
{  
 double S,K,r,v,t; //Declares Spot Price, Strike Price, Risk Free rate, Volatility and Time to expiry  
 char Opt\_type;   //Can be either 'C' ( for Call) or 'P' (for Put)  
 double \*pVol;     
 int Vol\_Smile\_flg=0;  
 double Opt\_Premium, Cal\_Opt\_Premium, Cal\_Opt\_Premium\_ATM\_Fixed,Opt\_Premium\_tweak, Cal\_Vol,Cal\_Vol\_ATM\_Fixed;  
 double S\_rel\_positivetweak, S\_rel\_negativetweak,S\_abs\_positivetweak,S\_abs\_negativetweak;  
 double delta,gamma,vega,theta,rho;  
 double delta1,gamma1,vega1,theta1,rho1;  
 char Job\_select;  
   
 /////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 //Taking User inputs for option pricing  
 /////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   
 std::cout << "What you would like to generate: Enter P for Option Pricing / Enter V for Vol Smile Fitting \n";  
 std::cin >> Job\_select;  
   
  if(Job\_select != 'P'){  
   if(Job\_select!='V'){  
    std::cout << "Incorrect entry.......exiting \n";  
    return 0;  
   }  
  }  
    
  if(Job\_select == 'V')  
  {  
   pVol=Vol\_Smile\_Fitting();  
   Vol\_Smile\_flg=1;  
  }  
   
 std::cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  
 std::cout<<"\n Enter Option Pricing Parameters\n";  
 std::cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";  
   
 std::cout << "\nEnter Spot Price: \n";  
 std::cin >> S;  
   
 std::cout << "\nEnter Strike of Option: \n";  
 std::cin >> K;  
   
 std::cout << "\nEnter Risk free Interest Rate (in fraction): \n";  
 std::cin >> r;  
   
 std::cout << "\nEnter Volatility of Underlying (in fraction): \n";  
 std::cin >> v;  
   
 std::cout << "\nEnter Option Expiry (in years): \n";  
 std::cin >> t;  
   
 std::cout << "\nWhich Option do you want to price: Enter C for Call/ P for Put \n";  
 std::cin >> Opt\_type;  
   
  if(Opt\_type != 'C'){  
   if(Opt\_type!='P'){  
   std::cout << "Incorrect Option type entry.......exiting \n";  
   return 0;  
   }  
  }   
    
  if(Vol\_Smile\_flg==1){  
   Cal\_Vol=\*(pVol+0)\*pow(K/S,2)+\*(pVol+1)\*K/S+\*(pVol+2);  
  }  
   if(Opt\_type=='P'){  
       Opt\_Premium = Put\_Opt\_Pricer(S,K,r,v,t) ;  
       if(Vol\_Smile\_flg==1){  
    Cal\_Opt\_Premium=Put\_Opt\_Pricer(S,K,r,Cal\_Vol,t) ;  
       }   
   }else if(Opt\_type=='C'){   
    Opt\_Premium = Call\_Opt\_Pricer(S,K,r,v,t);  
    if(Vol\_Smile\_flg==1){  
    Cal\_Opt\_Premium=Call\_Opt\_Pricer(S,K,r,Cal\_Vol,t) ;  
       }  
   }else{  
    std::cout << "Invalid Option Type \n";  
    std::cout << "Closing Program \n";  
    return 0;   
   }

 ///////////////////////////////////////////////////////////////////////////////////////  
 //Calculate Option Greeks using two different tweak methodology  
 //////////////////////////////////////////////////////////////////////////////////////  
   
 // Greeks Calculation Using Relative Tweaks  
  delta=Greeks(S\*1.01,K,r,v,t,Opt\_type)-Opt\_Premium;  
  S\_rel\_positivetweak=S\*1.01;  
  S\_rel\_negativetweak=S\*0.99;  
  gamma=(Greeks(S\_rel\_positivetweak,K,r,v,t,Opt\_type)-Greeks(S\_rel\_negativetweak,K,r,v,t,Opt\_type))/(S\_rel\_positivetweak-S\_rel\_negativetweak);  
 rho=Greeks(S,K,r\*1.01,v,t,Opt\_type)-Opt\_Premium;  
 vega=Greeks(S,K,r,v\*1.01,t,Opt\_type)-Opt\_Premium;  
 theta=Greeks(S,K,r,v,t\*(1-0.01),Opt\_type)-Opt\_Premium;

 // Greeks Calculation Using Absolute Tweaks  
 delta1=Greeks(S+1,K,r,v,t,Opt\_type)-Opt\_Premium;  
  S\_abs\_positivetweak=S+1;  
  S\_abs\_negativetweak=S-1;  
 gamma1=(Greeks(S\_abs\_positivetweak,K,r,v,t,Opt\_type)-Greeks(S\_abs\_negativetweak,K,r,v,t,Opt\_type))/(S\_abs\_positivetweak-S\_abs\_negativetweak);  
 rho1=Greeks(S,K,r+0.01,v,t,Opt\_type)-Opt\_Premium;  
 vega1=Greeks(S,K,r,v+.01,t,Opt\_type)-Opt\_Premium;  
 theta1=Greeks(S,K,r,v,t-(1/365),Opt\_type)-Opt\_Premium;   
   
 system("cls");  
   
 std::cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" <<std::endl;  
 std::cout << "Underlying Price:               " << S << std::endl;  
    std::cout << "Strike Price:                   " << K << std::endl;  
    std::cout << "Risk-Free Rate:                 " << r << std::endl;  
    std::cout << "Volatility:                     " << v << std::endl;  
    std::cout << "Maturity:                       " << t << std::endl;   
 std::cout << "Option Type:                    " << Opt\_type << std::endl;  
 std::cout << "Option Premium:                $" << Opt\_Premium << std::endl;  
   
  if(Vol\_Smile\_flg==1){  
   std::cout << "Calibrated Option Premium:       $"<< Cal\_Opt\_Premium << std::endl;  
   std::cout << "Calibrated Option Volatility:       "<< Cal\_Vol << std::endl;  
  }  
     
 std::cout << "\n\nOption Greeks are as below\n" << std::endl;  
 std::cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;  
 std::cout << "Relative Greeks:         " << std::endl;  
 std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" << std::endl;  
 std::cout << "Delta:              " << delta << std::endl;  
    std::cout << "Gamma:              " << gamma << std::endl;  
    std::cout << "Vega:               " << vega << std::endl;  
    std::cout << "Rho:                " << rho << std::endl;  
    std::cout << "Theta:              " << theta << std::endl;  
 std::cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;   
 std::cout << "Absolute Greeks:         " << std::endl;  
 std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" << std::endl;  
 std::cout << "Delta:              " << delta1 << std::endl;  
    std::cout << "Gamma:              " << gamma1 << std::endl;  
    std::cout << "Vega:               " << vega1 << std::endl;  
    std::cout << "Rho:                " << rho1 << std::endl;  
    std::cout << "Theta:              " << theta1 << std::endl;  
   
 return 0;  
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
//End of Equity Option Pricing and Risk Management module  
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////