**Spring 2020 FRE-GY 6883 Financial Computing**

**Midterm Exam**

Last Name: \_\_\_\_\_\_\_\_Guo\_\_\_\_\_\_\_\_First Name: \_\_\_\_\_\_\_Yihua (Sarah)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_NYUID: N16503778

**Q1: Complete the implementation of the member function Swap() and the main() function for the program SwapUtility to produce the expected result:**

|  |
| --- |
| SwapUtility.cpp |
| #include <iostream>  using namespace std;  class SwapUtility  {  private:  int\* const ptr1;  int\* const ptr2;  public:  SwapUtility() : ptr1(NULL), ptr2(NULL) {}  SwapUtility(int\* const ptr1\_, int\* const ptr2\_) : ptr1(ptr1\_), ptr2(ptr2\_) {} //cannot swap pointer, can only swap the value at location  **// Complete the Swap() function**  void Swap()  {  int temp = \*ptr1;  \*ptr1 = \*ptr2;  \*ptr2 = temp;  }  };  **// Complete the main function**  int main()  {  int a = 10, b = 20;  cout << "Before Swap()" << " a = " << a << " b = " << b << endl;  int\* ptr1 = &a;  int\* ptr2 = &b;  SwapUtility S(ptr1, ptr2);  S.Swap();  cout << "After Swap()" << " a = " << a << " b = " << b << endl;  return 0;  }  /\*  Before Swap() a = 10 b = 20  After Swap() a = 20 b = 10  \*/ |

**Q2. Complete NewBinModel.cpp to calculate asset price at each node to produce the following result.**

|  |
| --- |
| NewBinModel.cpp |
| #include <iostream>  #include <iomanip>  #include <cmath>  using namespace std;  int main()  { double u = 1.15125, d = 0.86862, r = 1.00545;  double s0 = 106.00;  for (int n = 0; n <= 8; n++)  { for (int i = 0; i <= n; i++)  { cout << "Asset Price at Binomaial Tree Node(" << n << "," << i << ") = " << fixed << setprecision(2);  cout << s0 \* pow(u, i) \* pow(d, n - i);  cout << endl;  }  }  return 0;  }  /\*  Asset Price at Binomaial Tree Node(0, 0) = 106.00  Asset Price at Binomaial Tree Node(1, 0) = 92.07  Asset Price at Binomaial Tree Node(1, 1) = 122.03  Asset Price at Binomaial Tree Node(2, 0) = 79.98  Asset Price at Binomaial Tree Node(2, 1) = 106.00  Asset Price at Binomaial Tree Node(2, 2) = 140.49  Asset Price at Binomaial Tree Node(3, 0) = 69.47  Asset Price at Binomaial Tree Node(3, 1) = 92.07  Asset Price at Binomaial Tree Node(3, 2) = 122.03  Asset Price at Binomaial Tree Node(3, 3) = 161.74  Asset Price at Binomaial Tree Node(4, 0) = 60.34  Asset Price at Binomaial Tree Node(4, 1) = 79.98  Asset Price at Binomaial Tree Node(4, 2) = 106.00  Asset Price at Binomaial Tree Node(4, 3) = 140.49  Asset Price at Binomaial Tree Node(4, 4) = 186.20  Asset Price at Binomaial Tree Node(5, 0) = 52.41  Asset Price at Binomaial Tree Node(5, 1) = 69.47  Asset Price at Binomaial Tree Node(5, 2) = 92.07  Asset Price at Binomaial Tree Node(5, 3) = 122.03  Asset Price at Binomaial Tree Node(5, 4) = 161.74  Asset Price at Binomaial Tree Node(5, 5) = 214.37  Asset Price at Binomaial Tree Node(6, 0) = 45.53  Asset Price at Binomaial Tree Node(6, 1) = 60.34  Asset Price at Binomaial Tree Node(6, 2) = 79.98  Asset Price at Binomaial Tree Node(6, 3) = 106.00  Asset Price at Binomaial Tree Node(6, 4) = 140.49  Asset Price at Binomaial Tree Node(6, 5) = 186.20  Asset Price at Binomaial Tree Node(6, 6) = 246.79  Asset Price at Binomaial Tree Node(7, 0) = 39.55  Asset Price at Binomaial Tree Node(7, 1) = 52.41  Asset Price at Binomaial Tree Node(7, 2) = 69.47  Asset Price at Binomaial Tree Node(7, 3) = 92.07  Asset Price at Binomaial Tree Node(7, 4) = 122.03  Asset Price at Binomaial Tree Node(7, 5) = 161.74  Asset Price at Binomaial Tree Node(7, 6) = 214.36  Asset Price at Binomaial Tree Node(7, 7) = 284.11  Asset Price at Binomaial Tree Node(8, 0) = 34.35  Asset Price at Binomaial Tree Node(8, 1) = 45.53  Asset Price at Binomaial Tree Node(8, 2) = 60.34  Asset Price at Binomaial Tree Node(8, 3) = 79.98  Asset Price at Binomaial Tree Node(8, 4) = 106.00  Asset Price at Binomaial Tree Node(8, 5) = 140.49  Asset Price at Binomaial Tree Node(8, 6) = 186.20  Asset Price at Binomaial Tree Node(8, 7) = 246.79  Asset Price at Binomaial Tree Node(8, 8) = 327.09  \*/ |

**Q3. Modify the above NewBinModel.cpp to build a Binomial Tree by using vector of vector of double.**

|  |
| --- |
| NewBinModel03.cpp |
| #include <vector>  #include<iostream>  #include <iomanip>  #include <cmath>  using namespace std;  int main()  {  double u = 1.15125, d = 0.86862, r = 1.00545;  double s0 = 106.00;  double aPrice = 0.0;  vector< vector<double> > price\_tree;  // compute asset price at all the nodes of the price\_tree  int index = 0;  cout << "The following is the populated price\_tree:" << endl;  for (int n = 0; n <= 8; n++)  {  Vector<double>prices;  For (int i = 0; i <= n; i++)  {  aPrice = s0 \* pow(u, i) \* pow(d, n - i);  prices.push\_back(aPrice);  }  prices.push\_back(aPrice);  price\_tree.resize(9);  price\_tree[n].resize(n + 1);  cout << fixed << setprecision(2);  cout << "step n = " << n << " ";  for (int i = 0; i <= n; i++)  {  price\_tree[n][i] = s0 \* pow(u, i) \* pow(d, n - i);  cout << price\_tree[n][i] << " ";  }  cout << endl;  }  return 0;}  /\* The following is the populated price\_tree:  step n = 0 106.00  step n = 1 92.07 122.03  step n = 2 79.98 106.00 140.49  step n = 3 69.47 92.07 122.03 161.74  step n = 4 60.34 79.98 106.00 140.49 186.20  step n = 5 52.41 69.47 92.07 122.03 161.74 214.37  step n = 6 45.53 60.34 79.98 106.00 140.49 186.20 246.79  step n = 7 39.55 52.41 69.47 92.07 122.03 161.74 214.36 284.11  step n = 8 34.35 45.53 60.34 79.98 106.00 140.49 186.20 246.79 327.09  \*/ |

|  |
| --- |
| **Q4-Q5 are based on the following classes:** |
| BinModel.h |
| #pragma once  class BinModel  {  private:  double S0;  double U;  double D;  double R;  public:  BinModel() : S0(0), U(0), D(0), R(0) {}  BinModel(double S0\_, double U\_, double D\_, double R\_) : S0(S0\_), U(U\_), D(D\_), R(R\_) {}  BinModel(const BinModel & binModel) : S0(binModel.S0), U(binModel.U), D(binModel.D), R(binModel.R) {}  ~BinModel() {}  double GetS0() const { return S0; }  double GetU() const { return U; }  double GetD() const { return D; }  double GetR() const { return R; }  double RiskNeutProb() const;  double CalculateAssetPrice(int n, int i) const;  void UpdateBinModel(double S0\_, double U\_, double D\_, double R\_);  }; |
| Options.h |
| #pragma once  #include "BinModel.h"  class Option  {  private:  Option() :N(0), K1(0), K2(0.0) {}  Option(const Option& option) : N(option.N), K1(option.K1), K2(option.K2) {}  protected:  int N;  double K1;  double K2;  public:  Option(int N\_, double K1\_, double K2\_) : N(N\_), K1(K1\_), K2(K2\_) {}  int GetN() const { return N; }  virtual double Payoff(double z) const = 0;  virtual ~Option() = 0;  };  class Call : public Option  {  public:  Call(int N\_, double K1\_) :Option(N\_, K1\_, 0.0) {}  ~Call() {}  double Payoff(double z) const;  };  class DigitCall : public Call  {  public:  DigitCall(int N\_, double K1\_) :Call(N\_, K1\_) { }  ~DigitCall() { }  double Payoff(double z) const;  };  class Put : public Option  {  public:  Put(int N\_, double K1\_) :Option(N\_, K1\_, 0.0) {}  ~Put() {}  double Payoff(double z) const;  };  class DigitPut : public Put  {  public:  DigitPut(int N\_, double K1\_) :Put(N\_, K1\_) { }  ~DigitPut() { }  double Payoff(double z) const;  };  class OptionCalculation  {  private:  Option\* pOption;  OptionCalculation() : pOption(0) {}  OptionCalculation(const OptionCalculation& optionCalculation) :pOption(optionCalculation.pOption) {}  public:  OptionCalculation(Option\* pOption\_) :pOption(pOption\_) {}  ~OptionCalculation() {}  double PriceByCRR(const BinModel& rModel);  double PriceBySnell(const BinModel& rModel);  }; |
| BinModel.cpp |
| #include "BinModel.h"  #include <cmath>  double BinModel::RiskNeutProb() const  {  return (R - D) / (U - D);  }  double BinModel::CalculateAssetPrice(int n, int i) const  {  // Codes are hidden due to Question 2  }  void BinModel::UpdateBinModel(double S0\_, double U\_, double D\_, double R\_)  {  S0 = S0\_;  U = U\_;  D = D\_;  R = R\_;  } |
| Option.cpp |
| #include "BinModel.h"  #include "Options.h"  #include <cmath>  #include <vector>  using namespace std;  Option::~Option() {}  double Call::Payoff(double z) const  {  if (z > K1) return z - K1;  return 0.0;  }  double DigitCall::Payoff(double z) const  {  if (z > K1) return 1.0;  return 0.0;  }  double Put::Payoff(double z) const  {  if (z < K1) return K1 - z;  return 0.0;  }  double DigitPut::Payoff(double z) const  {  if (z < K1) return 1.0;  return 0.0;  } |

**Q4. Complete the implementation the member functions PriceByCRR() and PriceBySnell() for the above class OptionCalculation.**

|  |
| --- |
| double OptionCalculation::PriceByCRR(const BinModel& binModel)  {  double q = binModel.RiskNeutProb();  int N = pOption->GetN();  vector<double> optionPrices(N + 1);  for (int i = 0; i <= N; i++)  optionPrices[i] = pOption->Payoff(binModel.CalculateAssetPrice(N, i)); //pOption is base class pointer  for (int n = N - 1; n >= 0; n--)  {  for (int i = 0; i <= n; i++)  optionPrices[i] = (q \* optionPrices[i + 1] + (1 - q) \* optionPrices[i]) / (1 + binModel.GetR());  }  return optionPrices[0];  } |
| double OptionCalculation::PriceBySnell(const BinModel& binModel)  {  double q = binModel.RiskNeutProb();  int N = pOption->GetN();  vector<double> optionPrices(N + 1);  double Payoff = 0.0;  for (int i = 0; i <= N; i++)  {  optionPrices[i] = pOption->Payoff(binModel.CalculateAssetPrice(N, i));  }  for (int n = N - 1; n >= 0; n--)  {  for (int i = 0; i <= n; i++)  {  optionPrices[i] = (q \* optionPrices[i + 1] + (1 - q) \* optionPrices[i]) / (1 + binModel.GetR());  Payoff = pOption->Payoff(binModel.CalculateAssetPrice(N, i));  if (Payoff > optionPrices[i]) optionPrices[i] = Payoff;  }  }  return optionPrices[0];  } |

**Q5. Complete the implementation of the main() function to produce the expected the results show as the following.**

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include "BinModel.h"  #include "Options.h"  using namespace std;  int main()  {  int N = 8;  double S0 = 106.00, K = 100.00;  double U = 0.15125, D = -0.13138, R = 0.00545;  BinModel Model(S0, U, D, R);  Option\* pCall = new Call(N, K);  Option\* pPut = new Put(N, K);  Option\* pDigitCall = new DigitCall(N, K);  Option\* pDigitPut = new DigitPut(N, K);  OptionCalculation O1(pCall); base class pointer points to current object  OptionCalculation O2(pPut);  OptionCalculation O3(pDigitCall);  OptionCalculation O4(pDigitPut);  cout << "American call option price = " << O1.PriceBySnell(Model) << endl;  cout << "American put option price =  " << O2.PriceBySnell(Model) << endl;  cout << "European digit call option price = " << O3.PriceByCRR(Model) << endl;  cout << "European digit put option price = " << O4.PriceByCRR(Model) << endl;  delete pCall;  pCall = NULL;  delete pPut;  pPut = NULL;  delete pDigitCall;  pDigitCall = NULL;  delete pDigitPut;  pDigitPut = NULL;  return 0;  }  /\*  American call option price = 21.68  American put option price = 11.72  European digit call option price = 0.58  European digit put option price = 0.38  \*/ |
| **Q6-Q8 are based on the following classes and independent functions:** |
| Solver03.h |
| #pragma once  template<typename Function>  double SolveByBisect(Function\* Fct, double Tgt, double LEnd, double REnd, double Acc)  { double left=LEnd, right=REnd, mid=(left+right)/2;  double y\_left=Fct->Value(left)-Tgt, y\_mid=Fct->Value(mid)-Tgt;  while (mid-left>Acc)  {  if ((y\_left>0 && y\_mid>0)||(y\_left<0 && y\_mid<0))  {left=mid; y\_left=y\_mid;}  else right=mid;  mid=(left+right)/2;  y\_mid=Fct->Value(mid)-Tgt;  }  return mid;  }  template<typename Function>  double SolveByNR(Function\* Fct, double Tgt, double Guess, double Acc)  { double x\_prev=Guess;  double x\_next=x\_prev -(Fct->Value(x\_prev)-Tgt)/Fct->Deriv(x\_prev);  while (x\_next-x\_prev>Acc || x\_prev-x\_next>Acc)  {  x\_prev=x\_next;  x\_next=x\_prev-(Fct->Value(x\_prev)-Tgt)/Fct->Deriv(x\_prev);  }  return x\_next;  } |
| Bond.cpp |
| #include "Solver03.h"  #include <map>  #include <vector>  #include <cmath>  #include <iostream>  #include <iomanip>  using namespace std;  class Bond  {  private:  double F; //face value  int T; //maturity time  double yield; //yield is private, needs to call the setYield function to update yield each time  map<int, double> couponMap;  public:  Bond(double F\_, int T\_, const vector<double>& C\_, const vector<int>& t\_):F(F\_),T(T\_),yield(0.0)  {  // Asssume vector C\_ and t\_ have same size  for (unsigned int i = 0; i < C\_.size(); i++)  couponMap[t\_[i]] = C\_[i];  }  double BondValue();  double BondDeriv();  double GetF() const { return F; }  int GetT() const { return T; }  double GetYield() const { return yield; }  void SetYield(double y) { yield = y; }  map<int, double>& GetCouponMap() { return couponMap; }  }; |

**Q6. Complete the implementation of BondValue() and BondDeriv() for the Bond class with the same formula we used in 2nd homework assignment for Nonlinear Solvers.**

|  |
| --- |
| double Bond::BondValue()  {  double P = 0;  for (map<int, double>::iterator it = couponMap.begin(); it != couponMap.end(); ++it)  P += it->second \* exp(-yield \* it->first);  P += F \* exp(-yield \* T);  return P;  }  double Bond::BondDeriv()  {  double D = 0;  for (map<int, double>::iterator it = couponMap.begin(); it != couponMap.end(); ++it)  D += -it->second \* it->first \* exp(-yield \* it->first);  D += -F \* T \* exp(-yield \* T);  return D;  } |

**Q7. Complete the implementation of the following class Intermediary, as well as the corresponding codes in main() to use it.**

|  |
| --- |
| class Intermediary  {  private:  Bond \* ptr;  public:  Intermediary(Bond\* ptr\_) : ptr(ptr\_) {}  **double Value(double y)**  **{**  ptr->SetYield(y);  return ptr->BondValue();  **}**  **double Deriv(double y)**  **{**  ptr->SetYield(y);  return ptr->BondDeriv();**}**  }; |
| int main()  {  double F=100.0; //face value  int T=3; //maturity time  vector<double> C; //coupons  C.push\_back(1.2); C.push\_back(1.2); C.push\_back(1.2);  vector<int> t; //coupon times  t.push\_back(1); t.push\_back(2); t.push\_back(3);  Bond MyBond(F,T,C,t);  double P=98.56;  double Acc=0.0001;  double y;  double LEnd = 0.0;  double REnd = 1.0;    **// Calculate the bond yield by bisection method**  Intermediary intermed(&MyBond);  y = SolveByBisect<Intermediary>(&intermed, P, LEnd, REnd, Acc);  cout << "Yield by bisection method: " << y << endl; //should not print here  Mybond.setYield(y);    cout << setiosflags(ios::fixed) << setprecision(4);  cout << "P = " << P << endl;  cout << MyBond << endl;  return 0;  }  /\*  P = 98.5600  F = 100.0000  T = 3  Year 1 Coupon: 1.2000  Year 2 Coupon: 1.2000  Year 3 Coupon: 1.2000  yield = 0.0168  \*/ |

**Q8. Complete the following operator overloading to print MyBond object in main() function shown above.**

|  |
| --- |
| ostream& operator << (ostream& out, Bond& bond)  {  out << "F = " << bond.GetF() << endl;  out << "T = " << bond.GetT() << endl;  map<int, double>& couponMap = bond.GetCouponMap();  for (map<int, double>::iterator itr = couponMap.begin(); itr != couponMap.end(); ++itr)  {  out << "Year " << (\*itr).first << " Coupon: " << (\*itr).second << endl;  }  out << endl;  out << "yield = " << bond.GetYield() << endl;  return out;  } |

**Q9. Complete the following implementation of calculating of delta and gamma based on Monto Carlo Simulation.**









|  |
| --- |
| void Rescale(SamplePath& S, double x)  { int m=S.size();  for (int j=0; j<m; j++) S[j] = x\*S[j];  }  double PathDepOption::PriceByMC(BSModel & Model, long N, double epsilon)  { double H=0.0, Hsq=0.0, Heps=0.0, Hmeps=0.0;  SamplePath S(m);  for(long i=0; i<N; i++)  { Model.GenerateSamplePath(T,m,S);  H = (i\*H + Payoff(S))/(i+1.0);  Hsq = (i\*Hsq + pow(Payoff(S),2.0))/(i+1.0);  Rescale(S,1.0+epsilon);  **// Complete the implementation of Heps and Hmeps**  Heps = (i / (i + 1.0)) \* Heps + Payoff(S) / (i + 1.0);  Rescale(S, (1.0 - epsilon)/(1.0 + epsilon));  Hmeps = (i / (i + 1.0)) \* Hmeps + Payoff(S) / (i + 1.0);  }  Price = exp(-Model.r\*T)\*H;  PricingError = exp(-Model.r\*T)\*sqrt(Hsq-H\*H)/sqrt(N-1.0);  **// Complete the implementation of delta and gamma. Assume delta and gamma are already declared.**  delta = exp(-Model.r \* T) \* (Heps - H) / (Model.S0 \* epsilon);  gamma = exp(-Model.r \* T) \* (Heps - 2 \* H + Hmeps) / (epsilon \* Model.S0 \* epsilon \* Model.S0);  return Price;  }  double ArthmAsianCall::Payoff(SamplePath& S)  {  double Ave=0.0;  for (int k=0; k<m; k++)  { Ave=(k\*Ave+S[k])/(k+1.0); }  if (Ave<K) return 0.0;  return Ave-K;  } |

**Q10. Complete the member functions PriceByBSFormula() and DeltaByBSFormula() of the GmtrAsianCall for the following program in order to calculate option price and delta for Arithmetic Asian Call based on Variance Reduction Monto Carlo procedure.**

The delta of the geometric Asian call could be calculated as:

****

|  |
| --- |
| **MCModel.h** |
| #pragma once  #include <vector>  #include <cstdlib>  #include <ctime>  using namespace std;  typedef vector<double> SamplePath;  class MCModel  {  private:  double S0, r, sigma;  public:  MCModel():S0(0.0), r(0.0), sigma(0.0) {}  MCModel(double S0\_, double r\_, double sigma\_):S0(S0\_), r(r\_), sigma(sigma\_)  {  srand((unsigned)time(NULL));  }  void GenerateSamplePath(double T, int m, SamplePath& S) const;  double GetS0() const { return S0; }  double GetR() const { return r; }  double GetSigma() const { return sigma; }  void SetS0(double S0\_) { S0 = S0\_; }  void SetR(double r\_) { r = r\_; }  void SetSigma(double sigma\_) { sigma = sigma\_; }  }; |
| **MCModel.cpp** |
| #include "MCModel.h"  #include <cmath>  const double pi=4.0\*atan(1.0);  double Gauss()  {  double U1 = (rand()+1.0)/(RAND\_MAX+1.0);  double U2 = (rand()+1.0)/(RAND\_MAX+1.0);  return sqrt(-2.0\*log(U1)) \* cos(2.0\*pi\*U2);  }  void MCModel::GenerateSamplePath(double T, int m, SamplePath& S) const  {  double St = S0;  for(int k=0; k<m; k++)  {  S[k]= St\*exp((r-sigma\*sigma\*0.5)\*(T/m)+sigma\*sqrt(T/m)\*Gauss());  St=S[k];  }  } |
| **Option.h** |
| #pragma once  #include "MCModel.h"  class Option  {  protected:  double T, K;  public:  virtual ~Option() {}  virtual double PriceByBSFormula(const MCModel& Model) { return 0.0; }  virtual double DeltaByBSFormula(const MCModel& Model) { return 0.0; }  virtual double Payoff(SamplePath& S) { return 0.0; }  }; |
| **EurCall.h** |
| #pragma once  #include "Option.h"  class EurCall : public virtual Option  {  private:  double d\_plus(const MCModel& Model);  double d\_minus(const MCModel& Model);  EurCall() : Option() {}  public:  EurCall(double T\_, double K\_)  {  T = T\_;  K = K\_;  }  double PriceByBSFormula(const MCModel& Model);  double VegaByBSFormula(const MCModel& Model);  double DeltaByBSFormula(const MCModel& Model); //we can override here  }; |
| **PathDepOption.h** |
| #pragma once  #include "Option.h"  class PathDepOption : public virtual Option  {  protected:  double Price, PricingError, delta;  int m;  PathDepOption() : m(0), Price(0.0), PricingError(0.0), delta(0.0) {}  public:  PathDepOption(double T\_, double K\_, int m\_) : m(m\_), Price(0.0), PricingError(0.0), delta(0.0)  {  T = T\_;  K = K\_;  }  virtual double Payoff(SamplePath& S) = 0;  double PriceByMC(const MCModel & Model, long N, double epsilon);  double PriceByVarRedMC(const MCModel & Model, long N, PathDepOption& CVOption, double epsilon);  double GetT() { return T; }  double GetPrice() { return Price; }  double GetPricingError() { return PricingError; }  double GetDelta() { return delta; }  };  class DifferenceOfOptions: public PathDepOption  {  private:  PathDepOption\* Ptr1;  PathDepOption\* Ptr2;  public:  DifferenceOfOptions(double T\_, double K\_, int m\_, PathDepOption\* Ptr1\_, PathDepOption\* Ptr2\_) :  PathDepOption(T\_, K\_, m\_), Ptr1(Ptr1\_), Ptr2(Ptr2\_)  { }  double Payoff(SamplePath& S)  {  return Ptr1->Payoff(S)-Ptr2->Payoff(S);  }  };  class ArthmAsianCall: public PathDepOption  {  public:  ArthmAsianCall(double T\_, double K\_, int m\_) :PathDepOption(T\_, K\_, m\_) {}  double Payoff(SamplePath& S);  }; |
| **GmtrAsianCall.h** |
| #pragma once  #include "EurCall.h"  #include "PathDepOption.h"  class GmtrAsianCall : public PathDepOption, public EurCall //multi inheritance  {  private:  double a, b;  void Calculate\_a\_b(const MCModel& Model);  public:  GmtrAsianCall(double T\_, double K\_, int m\_) : EurCall(T\_, K\_), PathDepOption(T\_, K\_, m\_), a(0.0), b(0.0) {}  double Payoff(SamplePath& S);  double PriceByBSFormula(const MCModel& Model);  double DeltaByBSFormula(const MCModel& Model);  }; |
| **EurCall.cpp** |
| #include "EurCall.h"  #include <cmath>  double N(double x)  {  double gamma = 0.2316419; double a1 = 0.319381530;  double a2 =-0.356563782; double a3 = 1.781477937;  double a4 =-1.821255978; double a5 = 1.330274429;  double pi = 4.0\*atan(1.0); double k = 1.0/(1.0+gamma\*x);  if (x>=0.0)  {  return 1.0-((((a5\*k+a4)\*k+a3)\*k+a2)\*k+a1)\*k\*exp(-x\*x/2.0)/sqrt(2.0\*pi);  }  else return 1.0-N(-x);  }  double EurCall::d\_plus(const MCModel& Model)  {  return (log(Model.GetS0()/K)+(Model.GetR()+0.5\*pow(Model.GetSigma(),2.0))\*T)/(Model.GetSigma()\*sqrt(T));  }  double EurCall::d\_minus(const MCModel& Model)  {  return d\_plus(Model)-Model.GetSigma()\*sqrt(T);  }  double EurCall::PriceByBSFormula(const MCModel& Model)  {  return Model.GetS0()\*N(d\_plus(Model))-K\*exp(-Model.GetR()\*T)\*N(d\_minus(Model));  }  double EurCall::VegaByBSFormula(const MCModel& Model)  {  double pi=4.0\*atan(1.0);  return Model.GetS0()\*exp(-d\_plus(Model)\*d\_plus(Model)/2)\*sqrt(T)/sqrt(2.0\*pi);  }  double EurCall::DeltaByBSFormula(const MCModel& Model)  {  return N(d\_plus(Model));  } |
| **PathDepOption.cpp** |
| #include "PathDepOption.h"  #include <cmath>  void Rescale(SamplePath& S, double epsilon)  {  int m=S.size();  for (int j=0; j<m; j++) S[j] = (1.0+epsilon)\*S[j];  }  double PathDepOption::PriceByMC(const MCModel & Model, long N, double epsilon)  {  double H=0.0, Hsq=0.0, Heps=0.0;  SamplePath S(m);  for(long i=0; i<N; i++)  {  Model.GenerateSamplePath(T,m,S);  H = (i\*H + Payoff(S))/(i+1.0);  Hsq = (i\*Hsq + pow(Payoff(S),2.0))/(i+1.0);  Rescale(S,epsilon);  Heps = (i\*Heps + Payoff(S))/(i+1.0);  }  Price = exp(-Model.GetR()\*T)\*H;  PricingError = exp(-Model.GetR()\*T)\*sqrt(Hsq-H\*H)/sqrt(N-1.0);  delta = exp(-Model.GetR()\*T)\*(Heps-H)/(Model.GetS0()\*epsilon);  return Price;  }  double PathDepOption::PriceByVarRedMC(const MCModel & Model, long N, PathDepOption& CVOption, double epsilon)  {  DifferenceOfOptions VarRedOpt(T,K,m,this,&CVOption);  Price = VarRedOpt.PriceByMC(Model,N,epsilon) + CVOption.PriceByBSFormula(Model);  delta = VarRedOpt.delta + CVOption.DeltaByBSFormula(Model);  PricingError = VarRedOpt.PricingError;  return Price;  }  double ArthmAsianCall::Payoff(SamplePath& S)  {  double Ave=0.0;  for (int k=0; k<m; k++) Ave=(k\*Ave+S[k])/(k+1.0);  if (Ave<K) return 0.0;  return Ave-K;  } |
| **main.cpp** |
| #include <iostream>  #include "PathDepOption.h"  #include "GmtrAsianCall.h"  using namespace std;  int main()  {  double S0=100.0, r=0.03, sigma=0.2;  MCModel Model(S0,r,sigma);  double T =1.0/12.0, K=100.0;  int m=30;  ArthmAsianCall Option(T,K,m);  GmtrAsianCall CVOption(T,K,m);  long N=30000;  double epsilon =0.001;  Option.PriceByVarRedMC(Model,N,CVOption,epsilon);  cout << "Arithmetic call price = " << Option.GetPrice() << endl  << "Error = " << Option.GetPricingError() << endl  << "delta = " << Option.GetDelta() << endl << endl;  Option.PriceByMC(Model,N,epsilon);  cout << "Price by direct MC = " << Option.GetPrice() << endl  << "Error = " << Option.GetPricingError() << endl  << "delta = " << Option.GetDelta() << endl;  return 0;  }  /\*  Arithmetic call price = 1.42594  Error = 0.000136447  delta = 0.520263  Price by direct MC = 1.42762  Error = 0.0120886  delta = 0.523995  \*/ |
| **GmtrAsianCall.cpp** |
| #include "GmtrAsianCall.h"  double GmtrAsianCall::Payoff(SamplePath& S)  {  double Prod = 1.0;  for (int i = 0; i < m; i++)  {  Prod = Prod \* S[i];  }  if (pow(Prod, 1.0 / m) < K) return 0.0;  return pow(Prod, 1.0 / m) - K;  }  void GmtrAsianCall::Calculate\_a\_b(const MCModel& Model)  {  a = exp(-Model.GetR() \* T) \* Model.GetS0() \* exp((m + 1.0) \* T / (2.0 \* m) \*  (Model.GetR() + Model.GetSigma() \* Model.GetSigma() \* ((2.0 \* m + 1.0) / (3.0 \* m) - 1.0) / 2.0));  b = Model.GetSigma() \* sqrt((m + 1.0) \* (2.0 \* m + 1.0) / (6.0 \* m \* m));  }  **// Complete the implementation of the member function, PriceByBSFormula**  double GmtrAsianCall::PriceByBSFormula(const MCModel& Model) cannot reference to const  {  MCMModel model;  model.SetS0(a);  model.SetSigma(b);  model.SetR(Model.GetR());  Price = EurCall::PriceByBSFormula(model);  Return Price;  Calculate\_a\_b(Model);  EurCall G(T,K);  Price = G.PriceByBSFormula(Model);  return Price;  }  **// Complete the implementation of the member function, DeltaByBSFormula**  double GmtrAsianCall::DeltaByBSFormula(const MCModel& Model)  {  Calculate\_a\_b(Model);  EurCall G(T,K);  MCMModel model;  model.SetS0(a);  model.SetSigma(b);  model.SetR(Model.GetR());  delta = G.DeltaByBSFormula(Model) \* a / Model.getS0();  return delta;  } |