Help

```
extern "C" {
#include  "
href../../../mod/doublehes1d/doublehes1d_std/doublehes1d_std_h_src.pdfhes1d_std.
#include "
href../../../common/enums_h_src.pdfenums.h"
}

#include "pnl/pnl_random.h"
#include <cmath>
using namespace std;

template <typename T> T log2(T x)
{
return log(x) / M_LOG2E;
}

// Brownian motion on a grid with time step T/2^l
static void Brownian(double* W,int l,double T,int generator)
{
int N = static_cast<int>(pow(2.0,l));
double sqrt_dt = sqrt(T/N);
for(int i=0;i<N;i++)
{
W[i] = sqrt_dt*pnl_rand_normal(generator);
}
}

// Swap the Brownian increments
static void Antithetic_Brownian(double* W,int l)
{
int N = static_cast<int>(pow(2.0,l-1));
double swap;
for(int i=0;i<N;i++)
{
swap = W[2*i];
W[2*i] = W[2*i+1];
W[2*i+1] = swap;
}
}
```

```
// Brownian motion on a grid with time step T/2^(l-1)
static void Coarse_Brownian(double* W,int l)
{
int N = static_cast<int>(pow(2.0,l-1));
for(int i=0;i<N;i++)
{
W[i] = W[2*i] + W[2*i+1];
}
}

template <typename T> static void Copy(T* W, T* B,int l)
{
int N = static_cast<int>(pow(2.0,l));
for(int i=0;i<N;i++)
{
B[i] = W[i];
}
}

// Bernoulli variables on a grid with time step T/2^l
static void Bernoulli(bool* Eta,int l,int generator)
{
int N = static_cast<int>(pow(2.0,l));
for(int i=0;i<N;i++)
{
Eta[i] = (pnl_rand_uni(generator)>0.5);
}
}

// Bernoulli variables on a grid with time step T/2^(l-1)
static void Coarse_Bernoulli(bool* Eta,int l)
{
int N = static_cast<int>(pow(2.0,l-1));
for(int i=0;i<N;i++)
{
Eta[i] = Eta[2*i];
}
}

static void Antithetic_Bernoulli(bool* Eta,int l)
```

```
{
int N = static_cast<int>(pow(2.0,l));
for(int i=0;i<N;i++)
{
Eta[i] = !(Eta[i]);
}
}

// *** Discretization *** //

/*
Heston Model:
dX_t = (r - \ divid - \ frac{1}{2} v_t)dt + \ sqrt{v_t} dW^1_t
dv_t = \ kappa (\ theta - v_t) dt + \ sigma \ rho \ sqrt{v_t} dW^1_t + \ sigma \
X_t = log(S_t)
*/


// Ninomiya Victoir scheme with time step T/2^l
static void Fine_NV_Scheme(double* W1,double* W2,bool* Eta,int l,double T,double
{
int N = static_cast<int>(pow(2.0,l));
double dt = T/double(N);
double ksi = theta - (sigma*sigma)/(4*kappa);
double omega = sqrt(1 - rho*rho);
for(int i=0;i<N;i++)
{
x += 0.5*(r - divid - 0.25*rho*sigma - 0.5*ksi)*dt + (1.0/(2*kappa))*(v - ksi)*(
v = (v - ksi)*exp(-0.5*kappa*dt)+ksi;
if(Eta[i])
{
x += sqrt(v)*W1[i] + 0.25*rho*sigma*W1[i]*W1[i];
v = (sqrt(v) + 0.5*sigma*rho*W1[i])*(sqrt(v) + 0.5*sigma*rho*W1[i]);
v = (sqrt(v) + 0.5*sigma*omega*W2[i])*(sqrt(v) + 0.5*sigma*omega*W2[i]);
}
else
{
v = (sqrt(v) + 0.5*sigma*omega*W2[i])*(sqrt(v) + 0.5*sigma*omega*W2[i]);
x += sqrt(v)*W1[i] + 0.25*rho*sigma*W1[i]*W1[i];
v = (sqrt(v) + 0.5*sigma*rho*W1[i])*(sqrt(v) + 0.5*sigma*rho*W1[i]);
}
```

```
x += 0.5*(r - divid - 0.25*rho*sigma - 0.5*ksi)*dt + (1.0/(2*kappa))*(v - ksi)*(
v = (v - ksi)*exp(-0.5*kappa*dt)+ksi;
}
}

// Ninomiya Victoir scheme with time step T/2^(l-1)
static void Coarse_NV_Scheme(double* W1,double* W2,bool* Eta,int l,double T,doub
{
Coarse_Bernoulli(Eta,l);
Coarse_Brownian(W1,l);
Coarse_Brownian(W2,l);
Fine_NV_Scheme(W1,W2,Eta,l-1,T,x,v,kappa,theta,sigma,r,divid,rho);
}

// Antithetic Ninomiya Victoir scheme with time step T/2^l
static void Antithetic_NV_Scheme(double* W1,double* W2,bool* Eta,int l,double T,
{
Antithetic_Brownian(W1,l);
Antithetic_Brownian(W2,l);
Fine_NV_Scheme(W1,W2,Eta,l,T,x,v,kappa,theta,sigma,r,divid,rho);
}

// *** Mathematical functions *** //
static void Mean_Variance(double &mean,double &variance,double S1,double S2,long
{
mean = S1/double(M);
variance = S2/double(M - 1) - (S1*S1)/(double(M)*double(M - 1));
}

// Min || Y - a e - b X ||^2 where e = (1,...,1)
static void Linear_Regression(double &intercept,double &slope,double* Y,double*
{
double Sx = 0;
double Sx2 = 0;
double Sy = 0;
double Sxy = 0;
for(int i=0;i<dim;i++)
{
Sx += X[i];
Sx2 += X[i]*X[i];
Sy += Y[i];
```

```
Sxy += X[i]*Y[i];
}
slope = (Sxy - Sx*Sy/double(dim))/(Sx2 - Sx*Sx/double(dim));
intercept = Sy/double(dim) - slope*Sx/double(dim);
}

// *** Payoff *** //
static double PutPayoff(double x,double K,double T,double r)
{
return double(exp(-r*T)*(K-exp(x))*(K>exp(x)));
}

// *** Monte Carlo *** //

static void Antithetic_Monte_Carlo_NV(double &mean,double &variance,int l,long M
{
int N = static_cast<int>(pow(2.0,l));
double* W1 = new double[N];
double* W2 = new double[N];
double* B1 = new double[N];
double* B2 = new double[N];
bool* Eta = new bool[N];
bool* _Eta = new bool[N];
double S1 = 0;
double S2 = 0;
double Res;
double xf,xa,xc,_xf,_xa,_xc;
double vf,va,vc,_vf,_va,_vc;
for(long i=0;i<M;i++)
{
xf = x;
vf = v;
xa = x;
va = v;
xc = x;
vc = v;
_xf = x;
_vf = v;
_xa = x;
_va = v;
_xc = x;
```

```
_vc = v;
Brownian(W1,l,T,generator);
Brownian(W2,l,T,generator);
Bernoulli(Eta,l,generator);
Copy(W1,B1,l);
Copy(W2,B2,l);
Copy(Eta,_Eta,l);
Antithetic_Bernoulli(_Eta,l);
Fine_NV_Scheme(W1,W2,Eta,l,T,xf,vf,kappa,theta,sigma,r,divid,rho);
Antithetic_NV_Scheme(W1,W2,Eta,l,T,xa,va,kappa,theta,sigma,r,divid,rho);
Coarse_NV_Scheme(W1,W2,Eta,l,T,xc,vc,kappa,theta,sigma,r,divid,rho);
Fine_NV_Scheme(B1,B2,_Eta,l,T,_xf,_vf,kappa,theta,sigma,r,divid,rho);
Antithetic_NV_Scheme(B1,B2,_Eta,l,T,_xa,_va,kappa,theta,sigma,r,divid,rho);
Coarse_NV_Scheme(B1,B2,_Eta,l,T,_xc,_vc,kappa,theta,sigma,r,divid,rho);
Res = 0.25*(PutPayoff(xf,K,T,r) + PutPayoff(xa,K,T,r) + PutPayoff(_xf,K,T,r) + P
S1 += Res;
S2 +=Res*Res;
}
delete[] W1;
delete[] W2;
delete[] B1;
delete[] B2;
delete[] Eta;
delete[] _Eta;
Mean_Variance(mean,variance,S1,S2,M);
}

void Crude_Monte_Carlo_NV(double &mean,double &variance,int l,long M,double K,do
{
int N = static_cast<int>(pow(2.0,l));
double* W1 = new double[N];
double* W2 = new double[N];
bool* Eta = new bool[N];
double S1 = 0;
double S2 = 0;
double Res;
double xf,vf;
for(long i=0;i<M;i++)
{
xf = x;
vf = v;
```

```
Brownian(W1,l,T,generator);
Brownian(W2,l,T,generator);
Bernoulli(Eta,l,generator);
Fine_NV_Scheme(W1,W2,Eta,l,T,xf,vf,kappa,theta,sigma,r,divid,rho);
Res = PutPayoff(xf,K,T,r);
S1 += Res;
S2 +=Res*Res;
}
delete[] W1;
delete[] W2;
delete[] Eta;
Mean_Variance(mean,variance,S1,S2,M);
}

// *** Optimal Parameters *** //
static void Intermediate_Step(double &c1,double &c2,double &alpha,double &beta,i
{
double mean,var;
double* L = new double[Lmax];
double* El = new double[Lmax];
double* Vl = new double[Lmax];
for(int l=1;l<=Lmax;l++)
{
L[l-1] = l;
Antithetic_Monte_Carlo_NV(mean,var,l,M,K,T,x,v,kappa,theta,sigma,r,divid,rho,gen
El[l-1] = log2(fabs(mean));
Vl[l-1] = log2(var);
}
Linear_Regression(c1,alpha,El,L,Lmax);
Linear_Regression(c2,beta,Vl,L,Lmax);
alpha *= -1.0;
beta *= -1.0;
c1 = exp(c1)/(1.0 - pow(2.0,alpha));
c2 = exp(c2);
delete[] L;
delete[] El;
delete[] Vl;
}

static void Last_Level(int &L,double epsilon,double c1,double alpha)
{
```

```
double _L = log2(sqrt(2.0)*fabs(c1)/epsilon)/alpha;
_L = _L*(_L>0);
L = static_cast<int>(_L + 1);
}

static void Optimal_Parameters(long* Ml,int L,double epsilon,double c1,double c2
{
double esp2 = epsilon*epsilon;
double p = 0.5*(1 - beta);
double sqrt_c2 = sqrt(c2);
double lambda = 5.0;
double C = sqrt(Var0) + sqrt(lambda)*sqrt_c2*((1 - pow(2.0,p*(L+1)))/(1 - pow(2.
Ml[0] = static_cast<long>((2.0/esp2)*C*sqrt(Var0));
for(int l=1;l<=L;l++)
{
Ml[l] = static_cast<long>((2.0/esp2)*C*sqrt_c2/sqrt(lambda*pow(2.0,l*(beta+1))))
}
}


extern "C"{

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2016+2) //The "#els
static int CHK_OPT(MC_AlgerbiJourdain)(void *Opt, void *Mod)
{
  return NONACTIVE;
}

int CALC(MC_AlgerbiJourdain)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


  int MCAlgerbiJourdain(double S,NumFunc_1  *p, double T, double r, double divid
{
  int flag_call;
  double K = p->Par[0].Val.V_PDOUBLE;
  int Lmax=4;
  long M=100000;
```

```
  double price_MC,variance_MC;
double x = log(S);
int L;
double c1,c2,alpha,beta,mean,var;
int l = 0;
Crude_Monte_Carlo_NV(mean,var,l,M,K,T,x,v,kappa,theta,sigma,r,divid,rho,generato
Intermediate_Step(c1,c2,alpha,beta,Lmax,M,K,T,x,v,kappa,theta,sigma,r,divid,rho,
Last_Level(L,epsilon,c1,alpha);
long* Ml = new long[L+1];
Optimal_Parameters(Ml,L,epsilon,c1,c2,alpha,beta,var);
price_MC = 0;
variance_MC = 0;
l = 0;
Crude_Monte_Carlo_NV(mean,var,l,Ml[0],K,T,x,v,kappa,theta,sigma,r,divid,rho,gene
price_MC += mean;
variance_MC += var/double(Ml[0]);
for(l=1; l<=L; l++)
{
Antithetic_Monte_Carlo_NV(mean,var,l,Ml[l],K,T,x,v,kappa,theta,sigma,r,divid,rho
price_MC += mean;
  variance_MC+= var/double(Ml[l]);
}

*ptprice=price_MC;
*st_dev=sqrt(variance_MC);

if ((p->Compute) == &Call)
    flag_call = 1;
  else
    flag_call = 0;

if (flag_call == 1)
    *ptprice = *ptprice - K * exp(-r * T) + S * exp(-divid * T);

delete[] Ml;

  return OK;
}

int CALC(MC_AlgerbiJourdain)(void *Opt, void *Mod, PricingMethod *Met)
{
```

```
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

  return MCAlgerbiJourdain(ptMod->S0.Val.V_PDOUBLE,
  ptOpt->PayOff.Val.V_NUMFUNC_1,
  ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
  r,
  divid, ptMod->Sigma0.Val.V_PDOUBLE
  , ptMod->MeanReversion.Val.V_PDOUBLE,
  ptMod->LongRunVariance.Val.V_PDOUBLE,
  ptMod->Sigma.Val.V_PDOUBLE,
  ptMod->Rho.Val.V_PDOUBLE,
  Met->Par[0].Val.V_ENUM.value,Met->Par[1].Val.V_PDOUBLE,
  &(Met->Res[0].Val.V_DOUBLE),
  &(Met->Res[1].Val.V_DOUBLE)
  );

}

static int CHK_OPT(MC_AlgerbiJourdain)(void *Opt, void *Mod)
{

  if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt
    return OK;

  return  WRONG;
}
#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  //int type_generator;
  if (Met->init == 0)
    {
      Met->init = 1;

      Met->Par[0].Val.V_ENUM.value = 0;
```

```
      Met->Par[0].Val.V_ENUM.members = &PremiaEnumMCRNGs;
  Met->Par[1].Val.V_DOUBLE = 0.01;
      Met->HelpFilenameHint = "mc_algerbijourdain";

    }
  return OK;
}

PricingMethod MET(MC_AlgerbiJourdain) =
{
  "MC_AlGerbi_Clement_Jourdain",
  {
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Error", DOUBLE, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_AlgerbiJourdain),
  { {"Price", DOUBLE, {100}, FORBID},
{"Std_dev", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_AlgerbiJourdain),
  CHK_mc,
  MET(Init)
};
}
```