Help

```
extern "C" {
#include  "
href../../../mod/doublehes1d/doublehes1d_std/doublehes1d_std_h_src.pdfhes1d_std.
#include "pnl/pnl_integration.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2017+2) //The "#els
static int CHK_OPT(TR_QUANTIZATION_CFG)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_QUANTIZATION_CFG)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else
////////////////////////////////////////////////////////////////
}

//using namespace std;
static double S0, r,q,V0,kappa,theta,rho,sigma,T; //parameters of the Heston mod
static double zi,mat,ubar,sinit,vinit; //variables needed for the Heston model
static int N; //variables for the quantization algorithm
static int call_or_put; //choice between put or call

static dcomplex charfunction (double u) //Implementation of the characteristic f
{

    dcomplex y1, y2, y3, y4, result;

    y1= Csqrt ( Cadd(   Cpow_real(CRadd(  CRmul(CI,rho*sigma*u), -kappa),2)  ,


    y2= Cdiv( Cadd(CRmul(CI,-rho*sigma*u),RCsub(kappa,y1)) , Cadd(CRmul(CI,-rho*
```

```
    y3=Cadd(CRmul(CI,(r-q)*u*mat),
            RCmul(kappa*theta/(sigma*sigma),
                  Csub(  CRmul(Cadd(CRmul(CI,-rho*sigma*u),RCsub(kappa,y1)),mat)
                       , RCmul(2,Clog( Cdiv(  RCsub(1, Cmul(y2,Cexp(CRmul(y1,-ma

    y4= Cdiv( Cmul(CRadd(Cminus(Cexp(CRmul(y1,-mat))),1) , Cadd(CRmul(CI,-rho*si

    result=Cexp( Cadd( Cadd( y3,CRmul(y4,vinit)) , CRmul(CI,u*log(sinit))      )

    return result;

}



static void initial_grid(PnlVect* x, int N){ //This function initialize the quan
    double a,b;

    //Values of a and b are heuristic
    if (mat>=0.3){
     a=0.95*S0;
     b=1.05*S0;
    }else{
        a=0.98*S0;
        b=1.02*S0;

    }

    for (int i=0;i<N;i++){
        LET(x,i)=a+(b-a)*i/(N-1);
    }

return;

}



static void xpm(PnlVect* x, PnlVect* xp,PnlVect* xm){ //Creation of xplus and xm
```

```
for (int i=0;i<N-1;i++) {
    LET(xp,i)=0.5*(GET(x,i)+GET(x,i+1));
    LET(xm,i+1)=0.5*(GET(x,i)+GET(x,i+1));
}

LET(xp,N-1)=0.5*(GET(x,N-1)+10*S0);
LET(xm,0)=0.5*(GET(x,0)+0.001);

return;
}

static double F1(double u, void* p){ //First function that we need to integrate

    return Creal(Cmul(charfunction(u), Cexp(CRmul(CI,-u*log(zi)))));

}

static double F2(double u, void* p){//Second function that we need to integrate

    return Creal( Cmul(  CRdiv(RCadd(1,CRmul(CI,u)),1+u*u) ,
                         Cmul(charfunction(u),        Cexp(CRmul(CI,-u*log(zi))))

}

static double F3(double u, void* p){//Third function that we need to integrate i

    return Creal( Cdiv(      Cmul(charfunction(u),        Cexp(CRmul(CI,-u*log(z

                         CRmul(CI,u)    )    )  ;

}



static void integrals(double *y1,double *y2,double *y3){ //This functions integr

    PnlFunc func1,func2,func3;
    char mode[]="simpson";
```

```
    func1.F=F1;
    func1.params=NULL;
    func2.F=F2;
    func2.params=NULL;
    func3.F=F3;
    func3.params=NULL;

    *y1=pnl_integration( &func1 ,1e-10,ubar,200,mode);
    *y2=pnl_integration( &func2 ,1e-10,ubar,200,mode);
    *y3=pnl_integration( &func3 ,1e-10,ubar,200,mode);

    return;

}



static void fill_vect_matr(PnlVect* y, PnlMat* A, PnlVect* x, PnlVect* xp, PnlVe
    PnlVect* CP=pnl_vect_create_from_zero(N);
    PnlVect* FP=pnl_vect_create_from_zero(N);
    PnlVect* GP=pnl_vect_create_from_zero(N);
    PnlVect* CM=pnl_vect_create_from_zero(N);
    PnlVect* FM=pnl_vect_create_from_zero(N);
    PnlVect* GM=pnl_vect_create_from_zero(N);
    double y1,y2,y3;


    //First step
    sinit=S0;
    vinit=V0;
    zi=GET(xm,0);
    integrals(&y1,&y2,&y3);



    LET(CM,0)=y1;
    LET(FM,0)=y2;
    LET(GM,0)=y3;
    //Intermediate steps
    for (int i=0;i<N-1;i++){
```

```
        zi=GET(xp,i);
    integrals(&y1,&y2,&y3);

        LET(CP,i)=y1;
LET(FP,i)=y2;
LET(GP,i)=y3;

LET(CM,i+1)=y1;
LET(FM,i+1)=y2;
LET(GM,i+1)=y3;


    }

    //Last step
    zi=GET(xp,N-1);
     integrals(&y1,&y2,&y3);
    LET(CP,N-1)=y1;
    LET(FP,N-1)=y2;
    LET(GP,N-1)=y3;
    //Filling

    for (int i=0;i<N;i++){

        LET(y,i)=GET(xp,i)*GET(FP,i) - GET(xm,i)*GET(FM,i) - GET(x,i)*(GET(GM,i)
        MLET(A,i,i)=0.5*(1 - GET(x,i)/GET(xp,i))*GET(CP,i) + GET(GP,i) - 0.5*(1

        if(i>0){
            MLET(A,i,i-1)=-0.5*(1 - GET(x,i)/GET(xm,i))*GET(CM,i);
             }
        if (i<N-1){
            MLET(A,i,i+1)= 0.5*(1 - GET(x,i)/GET(xp,i))*GET(CP,i);
        }
}



    pnl_vect_free(&CP);
    pnl_vect_free(&FP);
    pnl_vect_free(&GP);
    pnl_vect_free(&CM);
```

```
    pnl_vect_free(&FM);
    pnl_vect_free(&GM);

return;
}



static void fill_prob(PnlVect* prob, PnlVect* x, PnlVect* xp, PnlVect* xm){//Com

    PnlFunc prob_;
    char mode[]="simpson";
    prob_.F=F3;
    prob_.params=NULL;

    PnlVect* temp=pnl_vect_create_from_zero(N);

    zi=GET(xm,0);
    double t_tmp= pnl_integration( &prob_ ,1e-10,ubar,300,mode);
    zi=GET(xp,0);



    LET(temp,0)=pnl_integration( &prob_ ,1e-10,ubar,300,mode);
    LET(prob,0)=M_1_PI*(t_tmp - GET(temp,0));

    for(int i=1;i<N;i++){
        zi=GET(xp,i);
        LET(temp,i)=pnl_integration( &prob_ ,1e-10,ubar,300,mode);
        LET(prob,i)=M_1_PI*(-GET(temp,i)+GET(temp,i-1));
}

    pnl_vect_free(&temp);

return;
}


static void newton(int iter, PnlVect* x, PnlVect* prob){ //The quantization algo

    double toll=1e-2;
```

```
    if (mat<0.2){ubar=400;}else{ubar=100;} //This is euristic
    PnlVect* xp=pnl_vect_create_from_zero(N);
    PnlVect* xm=pnl_vect_create_from_zero(N);

    if (iter==0 || iter>=80){ //This can be used when implementing American opti
    initial_grid(x,N);
    }

    PnlMat* A=pnl_mat_create_from_zero(N,N);
    PnlVect* y=pnl_vect_create_from_zero(N);
    PnlVect* y_temp=pnl_vect_create_from_zero(N);

   //The Newton iteration in order to find the optimal quantization grid
    int iteration=1;
    do{
        xpm(x,xp,xm);
        fill_vect_matr( y,  A,  x,  xp,  xm);
        pnl_mat_syslin(y_temp,A,y);
        pnl_vect_minus_vect(x,y_temp);
        pnl_vect_qsort(x, 'i');
        iteration++;
        }while(pnl_vect_norm_two(y)>toll && iteration<=15);
    xpm(x,xp,xm);
    fill_prob(prob, x,xp,xm);


    pnl_vect_free(&xp);
    pnl_vect_free(&xm);
    pnl_vect_free(&y);
    pnl_vect_free(&y_temp);
    pnl_mat_free(&A);

return;

}



int TrQuantizationHeston(double r_fixed, double q_fixed, double s0_fixed,NumFunc
{
  double K;
```

```
  K = p->Par[0].Val.V_PDOUBLE;

  S0=s0_fixed;
  r=r_fixed;
  q=q_fixed;
  V0=v0_fixed;
  kappa=kappa_fixed;
  theta=theta_fixed;
  rho=rho_fixed;
  sigma=sigma_fixed;
  T=T_fixed;
  N=N_fixed;

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

   PnlVect* x=pnl_vect_create_from_zero(N);
   PnlVect* prob=pnl_vect_create_from_zero(N);


    mat=T;
    newton(0,x,prob);

    double res=0;
    for (int i=0;i<N;i++){
        if (call_or_put==0){
            res+=exp(-r*T)*MAX(GET(x,i)-K,0.0)*GET(prob,i);}
        else if(call_or_put==1){
            res+=exp(-r*T)*MAX(K-GET(x,i),0.0)*GET(prob,i);}
    }

  *ptprice=res;

   pnl_vect_free(&x);
   pnl_vect_free(&prob);

  return OK;
}
```

```
extern "C" {
int CALC(TR_QUANTIZATION_CFG)(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 TrQuantizationHeston(r,divid,ptMod->S0.Val.V_PDOUBLE,
                           ptOpt->PayOff.Val.V_NUMFUNC_1,
                           ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                          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_INT,
                           &(Met->Res[0].Val.V_DOUBLE)
  );

}

static int CHK_OPT(TR_QUANTIZATION_CFG)(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)
{
  if (Met->init == 0)
    {
  Met->HelpFilenameHint = "TR_QUANTIZATION_CFG";
  Met->Par[0].Val.V_INT = 50;
      Met->init = 1;
    }
```

```
  return OK;
}

PricingMethod MET(TR_QUANTIZATION_CFG) =
{
  "TR_QUANTIZATION_CallegaroFiorinGrasselli",
  {{"Size of the quantization grid", INT, {100}, ALLOW},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
  CALC(TR_QUANTIZATION_CFG),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(TR_QUANTIZATION_CFG),
  CHK_ok,
  MET(Init)
};
}
```