Help

```
#include "merhes1d_vol.h"
#include "pnl/pnl_complex.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2010+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(AP_MERHES_REALVAR)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_MERHES_REALVAR)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


static dcomplex cphi(double v, double s, double v0, double
    ka, double theta, double sigma, double gamma, double mu,
    double delta, double T);

/* ///////////////////////////////////////////////////// */
static int ap_hes_realvar(int ifCall, double v0,double ka,
    double theta,double sigma,double rhow,double r, double divid,
    double T, double Strike,  double gamma, double mu, double delta,
     double Spot, double parsigma, double parstep, int exp2,
    double *Price)
{

  double K, h, logstrikestep, A, odd, mval, vn, weight;
  double shift=parsigma;
  long int Nlimit, n;
  dcomplex fact, dzeta, xi, temp;
  double tt;
  double *y, *y_img, *k_arr;

  K=Strike;//p->Par[0].Val.V_DOUBLE;
  K=K*K*T/10000.0;
```

```
for(n=1,Nlimit=1;n<exp2+1;n++, Nlimit*=2);  //number of
  integral discretization steps
h=parstep;//step of integrtion

logstrikestep= 2*M_PI/Nlimit/h; //strike discretization
  step
A = Nlimit*h/2.0; // integration domain is (-A/2,A/2)
odd=-1.0; // to control Simpson's weights
//expectation of variance
mval= theta*T + ( v0 - theta )*( 1.0 - exp(-ka*T) )/ka +
  gamma*T*(mu*mu + delta*delta);// /T?

y= (double *)malloc((Nlimit)*sizeof(double));
y_img= (double *)malloc((Nlimit)*sizeof(double));
k_arr= (double *)malloc((Nlimit)*sizeof(double));

vn = -A;
//double weight = 0.5; //trapezoidal rule weights
weight = 1./3; //Simpson's rule weights

xi= Complex(shift, vn);
dzeta = Cdiv( RCmul( exp(-r*T), cphi(vn, shift, v0, ka,
  theta, sigma, gamma, mu, delta, T)),  Cmul( xi, xi) );///2.
  0;

y[0] = weight*dzeta.r;
y_img[0] = weight*dzeta.i;
k_arr[0] = K;

//price
for(n=1; n<Nlimit-1; n++){
  vn += h;
  //weight = 1; //trapezoidal rule weights
  odd*= -1.0; //weight = (weight<1) ? 4./3 : 2./3; //Simp
  son's rule weights
  temp=Complex(0.0, h*n*K);
xi= Complex(shift, vn);

  dzeta = Cdiv( RCmul( exp(-r*T), Cmul(Cexp(temp), cphi(
  vn, shift, v0, ka, theta, sigma, gamma, mu, delta, T))),
```

```
  Cmul( xi, xi) );

  //price
  y[n] = (1.0+odd*weight)*dzeta.r;
  y_img[n] = (1.0+odd*weight)*dzeta.i;
  k_arr[n]=K+n*logstrikestep;
}

vn += h;
//weight = 0.5; //trapezoidal rule weights
weight = 1.0/3.0;//Simpson's rule weights

temp=Complex(0.0, h*n*K);
xi= Complex(shift, vn);

dzeta = Cdiv( RCmul( exp(-r*T), Cmul(Cexp(temp), cphi(vn,
   shift, v0, ka, theta, sigma, gamma, mu, delta, T))),  Cm
  ul( xi, xi) );

y[Nlimit-1] = weight*dzeta.r;
y_img[Nlimit-1] = weight*dzeta.i;
k_arr[Nlimit-1] = K+(Nlimit-1)*logstrikestep;

pnl_ifft2(y,y_img,Nlimit);


if (ifCall)//((p->Compute)==&Call)
  {
    for(n=0;n<Nlimit-1;n++)
      {
        fact= CRdiv( CRmul(Cexp(CRmul( Complex(shift, -A)
  , k_arr[n] ) ), A), M_PI);
        tt=y[n];
        y[n]=fact.r*y[n]-fact.i*y_img[n] + exp(-r*T)*(mv
  al-k_arr[n]);
        y_img[n]=fact.r*y_img[n]+fact.i*tt;
        y[n]=y[n]>0?sqrt(y[n]/T)*100.0:-1;
        k_arr[n]=sqrt(k_arr[n]/T)*100.0;
      }
  }
else
```

```
    {
      for(n=0;n<Nlimit-1;n++)
        {
          fact=CRdiv( CRmul(Cexp(CRmul( Complex(shift, -A),
     k_arr[n] ) ), A), M_PI);
          tt=y[n];
          y[n]=fact.r*y[n]-fact.i*y_img[n];
          y_img[n]=fact.r*y_img[n]+fact.i*tt;
          y[n]=y[n]>0?sqrt(y[n]/T)*100.0:-1;
          k_arr[n]=sqrt(k_arr[n]/T)*100.0;
        }
    }

  *Price = y[0];

  free(y);
  free(y_img);
  free(k_arr);

  return OK;
}

/*-------------------------------------------------------*/
static dcomplex cphi(double v, double s, double v0, double
    ka, double theta, double sigma, double gamma, double mu,
    double delta, double T)
{
  double ss, dede;
  dcomplex x, d, edt, divedt, aa, bb, val, divkd, cc;

  x= Complex(s, v);

  ss = sigma*sigma;
  dede = delta*delta;
  d = Csqrt( RCadd(ka*ka, RCmul(2.0*ss, x) ) );
  edt = Cexp( CRmul(d, -T) );
  divkd = RCdiv(ka, d);
  divedt = Cadd( Cadd(CONE, divkd) , Cmul( Csub( CONE, div
    kd ), edt ) );
  aa = RCmul(2.0*theta*ka/ss, Cadd( CRmul( RCsub(ka,d), T/2
    .0) , RCsub( log(2.0), Clog(divedt) ) ) );
```

```
  bb = RCmul(2.0*v0, Cmul( Cdiv(x, d), Cdiv( Csub(CONE, ed
    t), divedt) ) );
  divkd= Cadd( RCmul(2.0*dede, x) , CONE);
  cc = Cdiv( Cexp( Cdiv( RCmul(-mu*mu, x) , divkd ) ), Csqr
    t(divkd) );
  cc= RCmul( gamma*T, Csub(cc, CONE) );
  val = Cexp( Cadd( Csub(aa, bb), cc) );

  return val;

}
/*----------------------------------------------------------
    -*/


int CALC(AP_MERHES_REALVAR)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r, divid, strike, spot;
  NumFunc_1 *p;
  int ifCall;

  r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
  divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
  p=ptOpt->PayOff.Val.V_NUMFUNC_1;
  strike=p->Par[0].Val.V_DOUBLE;
  spot=ptMod->S0.Val.V_DOUBLE;
  ifCall=((p->Compute)==&Call);

  return ap_hes_realvar(ifCall,
                        ptMod->Sigma0.Val.V_PDOUBLE,
                        ptMod->MeanReversion.hal.V_PDOUBLE,
                        ptMod->LongRunVariance.Val.V_PDOUB
    LE,
                        ptMod->Sigma.Val.V_PDOUBLE,
                        ptMod->Rho.Val.V_PDOUBLE,
                        r,divid,
                        ptOpt->Maturity.Val.V_DATE-ptMod->
    T.Val.V_DATE,
```

```
                        strike,
      ptMod->Lambda.Val.V_PDOUBLE,
            ptMod->Mean.Val.V_DOUBLE,
            ptMod->Variance.Val.V_PDOUBLE, spot,
                        Met->Par[0].Val.V_DOUBLE, Met->Par[
    1].Val.V_RGDOUBLE, Met->Par[2].Val.V_INT,
                        &(Met->Res[0].Val.V_DOUBLE));

}

static int CHK_OPT(AP_MERHES_REALVAR)(void *Opt, void *Mod)
{
  if ((strcmp( ((Option*)Opt)->Name,"CallRealVarEuro")==0 )
    ||strcmp( ((Option*)Opt)->Name,"PutRealVarEuro")==0 )
    return OK;

  return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  static int first=1;

  if (first)
    {
      Met->Par[0].Val.V_DOUBLE=10.0;
      Met->Par[1].Val.V_RGDOUBLE=0.5;
      Met->Par[2].Val.V_INT=12;

      first=0;
    }

  return OK;
}

PricingMethod MET(AP_MERHES_REALVAR)=
{
  "AP_MERHES_REALVAR",
  {   {"Shifting parameter for Laplace transform:", DOUBLE,
    {100},ALLOW    },
```

```
      {"Step of discretization for Laplace transform: ", RG
    DOUBLE,{100},ALLOW    },
      {"The log of Nb of points for Laplace transform",
    INT,{10},ALLOW    },
      {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(AP_MERHES_REALVAR),
  {   {"Price, in annual volatility points", DOUBLE,{100},
    FORBID},
      {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(AP_MERHES_REALVAR),
  CHK_ok ,
  MET(Init)
} ;

/*////////////////////////////////////*/
//}
```

References