Help

```
#include  "hes1d_std.h"
#include <pnl/pnl_mathtools.h>
#include <pnl/pnl_root.h>

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2012+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_Asymptotics_ImpliedVolatility)(void *
    Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_Asymptotics_ImpliedVolatility)(void*Opt,void *
    Mod,PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

//Large-Time Implied Volatility
//A Rate function
static double Vfun(double kappa, double theta, double sigma
    , double rho,double v0, double p)
{
double res;
res = kappa*theta*( kappa - rho*sigma*p-
sqrt(  SQR(kappa - rho*sigma*p)  + SQR(sigma)*(p-SQR(p) ) )

)/SQR(sigma);

return res;
}
/* static double VDerivee(double kappa, double theta,
    double sigma, double rho,double v0, double p)
 * {
 * double res;
 * res = kappa*theta*( - rho*sigma + ( rho*sigma*(kappa -
    rho*sigma*p) +SQR(sigma)*(p-0.5) )/sqrt(  SQR(kappa - rho*
    sigma*p)  + SQR(sigma)*(p-SQR(p) ) )
```

```
 * )/SQR(sigma);
 *
 * return res;
 * } */

static  double VSeconde(double kappa, double theta, double
    sigma, double rho,double v0, double p)
{
double res;
res = kappa*theta*( SQR(sigma)*(1.0-rho*rho)/sqrt(  SQR(ka
    ppa - rho*sigma*p)  + SQR(sigma)*(p-SQR(p) ) )  +
SQR( rho*sigma*(kappa - rho*sigma*p) +SQR(sigma)*(p-0.5) )/
    CUB(sqrt(  SQR(kappa - rho*sigma*p)  + SQR(sigma)*(p-SQR(p)
    ) ) )
)/SQR(sigma);

return res;
}

//P* gives ths Legendre Transform of V: It is defined as th
    e unique solution of V' (p* (x) ) = x
static double pstar(double kappa, double theta, double si
    gma, double rho,double v0, double x)
{
double res;
res = sigma - 2.0*kappa*rho + ( kappa*theta*rho + x*sigma)*

sqrt( (SQR(sigma) + SQR(2.0*kappa) -4.0*kappa*rho*sigma)/ (
     SQR(x*sigma)  + 2.0*x*kappa*theta*rho*sigma + SQR(kappa*
    theta)  ) );
res /= 2.0*sigma*(1 - rho*rho);

return res;
}

// V* (x) = x p*(x) - V(p*(x))
static double Vstar(double kappa, double theta, double si
    gma, double rho,double v0, double x)
{
double p = pstar( kappa,  theta,  sigma,  rho, v0,  x) ;
```

```
 return (x*p - Vfun( kappa,  theta,  sigma,  rho, v0,  p));

}

//The functions ABS and A are needed for the second term of
     the expansion of the implied volatility with respect to
    the maturity
static  double A_BS(double x, double sigma, double a)
{
double res;
if(x != SQR(sigma)/2.0 && x != -SQR(sigma)/2.0 )
res = exp( a*(SQR( 2.0*x/SQR(sigma) ) - 1.0 )/8.0 )* CUB(si
    gma)/(  SQR(x) - SQR( SQR(sigma)/2.0 ) ) ;
else
res = (a/2.0 - 1.0)/sigma ;

return res;
}

static double Afunc(double kappa, double theta, double si
    gma, double rho,double v0, double x)
{
double px = pstar( kappa,  theta,  sigma,  rho, v0,  x);
double V = Vfun( kappa,  theta,  sigma,  rho, v0,  px);
double V2 =  VSeconde( kappa,  theta,  sigma,  rho, v0,  px
    );
double U, d;
double res;

 d = sqrt(  SQR(kappa - rho*sigma*px)  + SQR(sigma)*(px-SQ
    R(px) ) );

U = exp(2.0*kappa*theta* log(  2.0*d/(kappa - rho*sigma*px
    +d ) )/SQR(sigma)  ) * exp( v0*V/(kappa*theta) );

 res = U/( sqrt(V2)* px*(px-1.0) );

return res;

}
```

```
int ApIVAsymptoticsHeston(double S0,NumFunc_1  *Payoff,
    double T, double r, double div,double v0,double kappa,double th
    eta,double sigma,double rho,double *implied_vol)
{
  double sigmaInfty, a1;
  double kappaBar = kappa - rho*sigma;
  double barTheta = kappa*theta/kappaBar;
  double Vx;
  double A,Abs;
  double x,K;

  K=Payoff->Par[0].Val.V_PDOUBLE;
  x = log(K*exp(-(r-div)*T)/S0)/T;
  if(x == 0)
    x = 0.001;


  Vx = Vstar( kappa,  theta,  sigma,  rho, v0,  x);
  pstar( kappa,  theta,  sigma,  rho, v0,  x);

  sigmaInfty = 2.0*( 2.0*Vx-x);

  if(x > -theta/2.0 && x< barTheta/2.0)
    sigmaInfty += 4.0*sqrt(  SQR(Vx) - Vx*x );
  else
    sigmaInfty -= 4.0*sqrt(  SQR(Vx) - Vx*x );

  Abs = A_BS(x, sigmaInfty,0);

  A = Afunc( kappa,  theta,  sigma,  rho, v0,  x);

  if(A*Abs >=0.0)
    a1 = 2.0* log( A/Abs )/( SQR( x/SQR(sigmaInfty) ) -1.0/
    4.0 );
  else
    a1 = 0.0;

  /* Implied Volatility*/
  *implied_vol=sqrt( ABS(  sigmaInfty  + a1/T));
```

```
  return OK;
}

int CALC(AP_Asymptotics_ImpliedVolatility)(void *Opt, void
    *Mod, PricingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r,divid;

  if(ptMod->Sigma.Val.V_PDOUBLE==0.0)
    {
      Fprintf(TOSCREEN,"BLACK-SHOLES MODEL{n{n{n");
      return WRONG;
    }
  else
    {
      r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
      divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);

      return ApIVAsymptoticsHeston(ptMod->S0.Val.V_PDOUBLE,
                                   ptOpt->PayOff.Val.V_
    NUMFUNC_1,
                                   ptOpt->Maturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,
                                   r,
                                   divid, ptMod->Sigma0.Val
    .V_PDOUBLE
                                   ,ptMod->MeanReversion.h
    al.V_PDOUBLE,
                                   ptMod->LongRunVariance.
    Val.V_PDOUBLE,
                                   ptMod->Sigma.Val.V_PDOUB
    LE,
                                   ptMod->Rho.Val.V_PDOUB
    LE,
                                   &(Met->Res[0].Val.V_
    DOUBLE)
        );
    }
```

```
}

static int CHK_OPT(AP_Asymptotics_ImpliedVolatility)(void *
    Opt, void *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"CallEuro")==0))

    return OK;
  return WRONG;
}

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

  return OK;
}

PricingMethod MET(AP_Asymptotics_ImpliedVolatility)=
{
  "AP_Asymptotics_ImpliedVolatility",
  {{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(AP_Asymptotics_ImpliedVolatility),
  {{"Asymptotics for Implied Volatility ",DOUBLE,{100},FORB
    ID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(AP_Asymptotics_ImpliedVolatility),
  CHK_ok,
  MET(Init)
};
```

References