Help

```
#include  "hescir1d_std.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(AP_MedvedevScaillet)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_MedvedevScaillet)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else
////////////////////////////////////////////////////////////////
static void fy0(double S, double K, double t, double v, double *result)
{
  * result = log(K / S) / v / sqrt(t);
}
static void fb1(double sigmav, double *result)
{
  * result = sigmav / 2.;
}
static void fav(double kv, double vbar, double b1, double v, double *result)
{
  *result = (kv * (vbar - pow(v, 2)) - pow(b1, 2)) / 2 / v;
}
static void fav1(double kv, double vbar, double b1, double v, double *result)
{
  * result = -(kv * vbar - pow(b1, 2)) / 2 / pow(v, 2) - kv / 2;
}
static void fav2(double kv, double vbar, double b1, double v, double *result)
{
  * result = (kv * vbar - pow(b1, 2)) / pow(v, 3);
}
static void ff(double r, double d, double *result)
{
  * result = r - d;
}
static void far(double kr, double rbar, double r, double *result)
```

```
{
  * result = kr * (rbar - r);
}
static void far1(double kr, double *result)
{
  * result = -kr;
}
static void fbr(double sigmar, double r, double *result)
{
  * result = sigmar * sqrt(r);
}
static void fbr1(double sigmar, double r, double *result)
{
  * result = sigmar / sqrt(r) / 2;
}
static void fbr2(double sigmar, double r, double *result)
{
  * result = -(1 / 4) * sigmar / pow(r, (3 / 2));
}

static void theta(double S, double K, double t, double v, double *result)
{
  * result = log(K / S) / v / sqrt(t);
}

//Coefficients and their derivatives
static void fC1(double N0, double n0, double K, double y, double v, double *resu
{
  * result = K * y * v / (N0 * y + n0);
}
static void fVC1(double N0, double n0, double K, double y, double *result)
{
  * result = K * y / (N0 * y + n0);
}
static void fC2(double N0, double n0, double C1, double K, double v, double f, d
{
  * result = 1 / 2.*(-N0 * C1 * pow(v, 2) + 2 * N0 * C1 * f - n0 * y * b1 * q1 *
}
static void fVC2(double N0, double n0, double C1, double K, double y, double v,
{
  * result = -(N0 * C1 + K * pow(y, 2) * v) / (N0 * pow(y, 2) + N0 + n0 * y);
```

```
}
static void fRC2(double N0, double n0, double C1, double y, double v, double *re
{
  * result = N0 * C1 / (v * (N0 * pow(y, 2) + N0 + n0 * y));
}
static void fVVC2(double N0, double n0, double C1, double K, double v, double y,
{
  * result = -(N0 * C1 + K * pow(y, 2) * v) / (v * (N0 * pow(y, 2) + N0 + n0 * y
}
static void fC3(double N0, double n0, double C1, double C2, double VC2, double R
{
  *result = 1 / 24.*(-3 * n0 * C1 * pow(v, 4) + 48 * N0 * y * b1 * q1 * VC2 * po
}
static void fVC3(double N0, double n0, double C1, double C2, double VC2, double
{
  *result = 1 / 24.*(-12 * n0 * av1 * pow(v, 2) * C1 - 9 * n0 * C1 * pow(v, 4) -
}
static void fRC3(double N0, double n0, double C1, double C2, double RC2, double
{
  *result = -1 / 2.*(-4 * n0 * br1 * q2 * pow(v, 2) * RC2 + 8 * N0 * y * b1 * q1
}
static void fVVC3(double N0, double n0, double C1, double C2, double VC2, double
{
  *result = 1 / 12.*(-9 * n0 * C1 * pow(v, 4) + 96 * N0 * y * b1 * q1 * VC2 * po
}
static void fVRC3(double N0, double n0, double C1, double C2, double RC2, double
{
  *result = -1 / 2.*(-4 * n0 * br1 * q2 * pow(v, 2) * RC2 + 8 * N0 * y * b1 * q1
}
static void fRRC3(double N0, double n0, double C1, double RC2, double br2, doubl
{
  *result = 1 / 2.*(-2 * n0 * C1 + 8 * N0 * y * RC2 * v + 4 * N0 * y * br2 * q2
}
static void fC4(double N0, double n0, double C1, double C2, double C3, double VC
{
  *result = -1 / 48.*(6 * n0 * pow(y, 5) * pow(b1, 2) * pow(q1, 2) * C1 * f + 12
}
static void fRC4(double N0, double n0, double C1, double C2, double C3, double V
{
  *result = -1 / 48.*(6 * n0 * pow(y, 5) * pow(b1, 2) * pow(q1, 2) * C1 * f + 12
}
```

```
static void fVC4(double N0, double n0, double C1, double C2, double C3, double V
{
  *result = 1 / 24.*(-12 * n0 * pow(y, 3) * pow(b1, 2) * pow(q1, 2) * C2 * v - 2
}
static void fC5(double N0, double n0, double C1, double C2, double C3, double C4
{
v, 4) * pow(b1, 2) *result=1344.0/*576.0*(-576.0*N0*pow(y,3)*pow(q1, 4) * VC2*br*pow(b1,2)*RC2
}


//Elements of the formula
static void fPn1(double C1, double *result)
{
  * result = C1;
}
static void fPN1(double C1, double z, double *result)
{
  * result = z * C1;
}
static void fPn2(double C1, double C2, double v, double b1, double q1, double z,
{
  *result = 1 / 2.*z * (2 * C2 * v + b1 * q1 * C1) / v;
}
static void fPN2(double C1, double C2, double v, double z, double f, double *res
{
  *result = 1 / 2.*(2 * C2 * v * pow(z, 2) + 2 * C2 * v + C1 * pow(v, 2) - 2 * C
}
static void fPn3(double C1, double C2, double C3, double VC2, double RC2, double
{
  *result = 1 / 24.*(24 * C3 * pow(v, 2) * pow(z, 2) + 48 * C3 * pow(v, 2) + 3 *
}
static void fPN3(double C1, double C2, double C3, double VC2, double RC2, double
{
  *result = -z * (-C3 * v * pow(z, 2) - 3 * C3 * v + 2 * br * q2 * v * RC2 + 2 *
}
static void fPn4(double C1, double C2, double C3, double C4, double VC2, double
{
  *result = 1 / 48.*z * (48 * ar * pow(v, 3) * RC2 - 48 * C2 * pow(v, 3) * r + 1
}
static void fPN4(double C1, double C2, double C3, double C4, double VC2, double
{
```

```
  *result = 1 / 4.*(-4 * av * v * C2 + 4 * av * pow(v, 2) * VC2 + 4 * ar * pow(v
}
static void fPn5(double C1, double C2, double C3, double C4, double C5, double V
{
, 4) * RRC3 + 900 * *result = 1 / 5760.*(2 * 11520 * - 1440 * pow(v, 4) * C3 + 31680 * pow(b1, 2) * pow(b1,
}
static void fPN5(double C1, double C2, double C3, double C4, double C5, double V
{
  *result = 1 / 4.*z * (8 * b1 * q1 * pow(v, 2) * av1 * C2 - 4 * b1 * q1 * pow(v
}

static void price_barrier_put(double y, double S, double K, double t, double v,
{
  double /*f, z, b1, av, av1, av2, q, ar, ar1, br, br1, br2, */n, N, n0, N0, C1,

  N0 = cdf_nor(y);
  n0 = pnl_normal_density(y);

  fC1(N0, n0, K, y, v, & C1);
  fVC1(N0, n0, K, y, & VC1);
  fC2(N0, n0, C1, K, v, f, y, b1, q1, & C2);
  fVC2(N0, n0, C1, K, y, v, & VC2);
  fRC2(N0, n0, C1, y, v, & RC2);
  fVVC2(N0, n0, C1, K, v, y,  & VVC2);
  fC3(N0, n0, C1, C2, VC2, RC2, b1, br, q1, q2, av, K, v, y, r, f, & C3);
  fVC3(N0, n0, C1, C2, VC2, VVC2, RC2, b1, br, q1, q2, av, av1, K, v, y, r, f, &
  fRC3(N0, n0, C1,  C2, RC2,  b1,  br1,  q1,  q2,  K,  v,  y,  r,  f,  & RC3);
  fVVC3(N0,  n0, C1,  C2,  VC2,  VVC2,  b1,  q1,  av2,  K,  v,  y,  r,  f, & VVC
  fVRC3(N0, n0, C1, C2, RC2, b1, br1, q1, q2, K, v, y, r, f, & VRC3);
  fRRC3(N0, n0, C1, RC2, br2, q2, K, v, y, r, f, & RRC3);
  fC4(N0, n0, C1, C2, C3, VC2, VC3, VVC2, RC2, RC3, b1, br, br1, q1, q2, av, av1
  fRC4(N0, n0, C1, C2, C3, VC2, VC3, VVC2, RC2, RC3, b1, br, br1, q1, q2, av, av
  fVC4(N0, n0, C1, C2, C3, VC2, VC3, VVC2, VVC3, VRC3, RC2, RC3, b1, br, br1, q1
  fC5(N0, n0, C1, C2, C3, C4, VC2, VC3, VC4, VVC2, VVC3, VRC3, RC2, RC3, RC4, RR

  fPn1(C1, & Pn1);
  fPN1(C1, z, & PN1);
  fPn2(C1, C2, v, b1, q1, z, & Pn2);
  fPN2(C1, C2, v, z, f, & PN2);
  fPn3(C1, C2, C3, VC2, RC2, r, v, z, f, av, b1, br, q1, q2, & Pn3);
  fPN3(C1, C2, C3, VC2, RC2, z, r, v, f, b1, br, q1, q2, & PN3);
```

```
  fPn4(C1, C2, C3, C4, VC2, VVC2, VC3, RC2, RC3, b1, br, br1, q1, q2, av, av1, a
  fPN4(C1, C2, C3, C4, VC2, VVC2, VC3, RC2, RC3, b1, br, br1, q1, q2, av, av1, a
  fPn5(C1, C2, C3, C4, C5, VC2, VVC2, VVC3, VC3, VC4, RC2, RC3, RC4, RRC3, VRC3,
  fPN5(C1, C2, C3, C4, C5, VC2, VVC2, VVC3, VC3, VC4, RC2, RC3, RC4, RRC3, VRC3,
  N = cdf_nor(z);
  n = pnl_normal_density(z);
  *result = (Pn1 * n + PN1 * N) * sqrt(t) + (Pn2 * n + PN2 * N) * t + (Pn3 * n +
}


static void pricing_american_put_three_factor_model(double S, double K, double t
{
  double f, z, b1, av, av1, av2, q = 0., ar, ar1, br, br1, br2, y, y0, step, pri
  int dir, search, i;
  fb1(sigmav, & b1);
  fav(kv, vbar, b1, v, & av);
  fav1(kv, vbar, b1, v, & av1);
  fav2(kv, vbar, b1, v, & av2);
  ff(r, d, & f);
  far(kr, rbar, r, & ar);
  far1(kr, & ar1);
  fbr(sigmar, r, & br);
  fbr1(sigmar, r, & br1);
  fbr2(sigmar, r, & br2);

  theta(S, K, t, v, & z);

  fy0(S, K, t, v, & y0);
  step = 1;
  y = y0;
  price0 = K - S;
  while (step >= 0.001)
    {
      dir = 1;
      search = 1;
      i = 0;
      while (search == 1 && i <= 1000)
        {
          i = i + 1;
          price_barrier_put(y + dir * step, S, K, t, v, r, d, kv,  vbar, sigmav,
          if (price1 > price0)
```

```
            {
              y = y + dir * step;
              result[3] = y;
              price0 = price1;
            }
          else
            {
              search = 0;
            }
        }
      step = step / 10;
      price_barrier_put(y - step, S, K, t, v, r, d, kv,  vbar, sigmav,  kr,  rba
      price_barrier_put(y + step, S, K, t, v, r, d, kv,  vbar, sigmav,  kr,  rba
      if (price2 > price1) dir = 1;
      else dir = -1;
    }
  result[3] = y; // Exercised Bound
  result[0] = price0; // American put price
  price_barrier_put(10000, S, K, t, v, r, d, kv,  vbar, sigmav,  kr,  rbar,  sig
  result[1] = price2; //European put price
  result[2] = result[0] - result[1]; // Early Exercise premium
}

int ApMedvedevScaillet(int am, double spot, NumFunc_1  *p, double maturity, doub
{
  double strike, price, delta, spot_inc, volatility;
  double Result[4];
  double inc = 0.0001;
  double dividend = 0.;
  volatility = sqrt(v0);
  strike = p->Par[0].Val.V_PDOUBLE;

  pricing_american_put_three_factor_model(spot, strike, maturity, volatility, in

  if (am)
    price = Result[0];
  else price = Result[1];
  //Price
  *ptprice = price;
```

```
  //Delta
  spot_inc = spot * (1 + inc);
  pricing_american_put_three_factor_model(spot_inc, strike, maturity, volatility
  if (am)
    delta = (Result[0] - price) * (spot * inc);
  else
    delta = (Result[1] - price) * (spot * inc);

  *ptdelta = delta;

  return OK;
}

int CALC(AP_MedvedevScaillet)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;


  return ApMedvedevScaillet(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
                            ptOpt->PayOff.Val.V_NUMFUNC_1,
                            ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                            ptMod->r0.Val.V_PDOUBLE
                            , ptMod->kr.Val.V_PDOUBLE,
                            ptMod->thetar.Val.V_PDOUBLE,
                            ptMod->Sigmar.Val.V_PDOUBLE,
                            ptMod->V0.Val.V_PDOUBLE
                            , ptMod->kV.Val.V_PDOUBLE,
                            ptMod->thetaV.Val.V_PDOUBLE,
                            ptMod->SigmaV.Val.V_PDOUBLE,
                            ptMod->RhoSr.Val.V_RGDOUBLE,
                            ptMod->RhoSV.Val.V_RGDOUBLE,
                            ptMod->RhorV.Val.V_RGDOUBLE,
                            &(Met->Res[0].Val.V_DOUBLE),
                            &(Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(AP_MedvedevScaillet)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "PutAmer") == 0) || (strcmp(((Option *)Opt)
    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_MedvedevScaillet) =
{
  "AP_MedvedevScaillet",
  {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(AP_MedvedevScaillet),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Delta", DOUBLE, {100}, FORBID} ,
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(AP_MedvedevScaillet),
  CHK_ok,
  MET(Init)
};
```