Help

```
#include  "hescir1d_std.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2011+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_GrzelakOosterlee)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
int CALC(AP_GrzelakOosterlee)(void*Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else



void static funcD1(dcomplex u, double lambda, double eta,
    dcomplex *result)
{
  if (lambda==0.0 || eta==0.0)  *result=CZERO;
  else  *result=Csqrt(RCadd(lambda*lambda, RCmul(2.*eta*
    eta, RCsub(1., Cmul(u, CI)))));
}
void static funcD2(dcomplex u, double gamma1, double rho12,
     double k, dcomplex *result)
{
  if (k==0.0 || gamma1==0.0)  *result=CZERO;
  else  *result=Csqrt(Csub(Cpow_real(CRsub(RCmul(gamma1*
    rho12, Cmul(CI,u)), k), 2.0), Cmul( CRsub(Cmul(u, CI), 1.),
     RCmul(gamma1*gamma1, Cmul(u,CI)))));
}
void static funcG1(double lambda, dcomplex D1, dcomplex *
    result)
{
  if (Creal(D1)==0.0 && Cimag(D1)==0.0)  *result=CZERO;
```

```
  else  *result=Cdiv(RCsub(lambda, D1), RCadd(lambda,D1))
    ;
}
void static funcG2(dcomplex u, double k, double gamma1,
    double rho12, dcomplex D2, dcomplex *result)
{
  if (Cimag(D2)==0.0 && Creal(D2)==0.0)  *result=CZERO;
  else  *result=Cdiv(Csub(RCsub(k, RCmul(gamma1*rho12, Cm
    ul(CI,u))), D2), Cadd(RCsub(k, RCmul(gamma1*rho12, Cmul(CI,
    u))), D2));
}
void static funcBx(dcomplex u, dcomplex *result)
{
  *result= Cmul (u, CI);
}
void static funcBr(dcomplex u, double tau, double lambda,
    double eta, dcomplex D1, dcomplex G1, dcomplex *result)
{
  if(eta==0.0)
  {
    if (lambda==0.0)  *result=RCmul(tau, CRsub(Cmul(u,
    CI), 1.0));
    else *result = RCmul((1.0-exp(-lambda*tau))/lambda,
    CRsub(Cmul(u, CI), 1.0));
  }
  else *result= Cmul( Cdiv (RCsub (1., Cexp(RCmul(-tau, D1
    ))), RCmul (eta*eta, RCsub (1., Cmul (G1, Cexp (RCmul(-ta
    u, D1)))))), RCsub (lambda, D1));
}
void static funcBsigma(dcomplex u, double tau, double k,
    double gamma1, double rho12,  dcomplex D2, dcomplex G2, dcompl
    ex *result)
{
  if (gamma1==0.0)
  {
    if (k==0.0)  *result = RCmul(-tau/2.0, Cadd(Cmul(u,
    u), Cmul(u, CI)));
    else  *result = RCmul((exp(-k*tau)-1.0)/(2.0*k), Cad
    d(Cmul(u, u), Cmul(u, CI)));
  }
  else  *result=Cmul (Cdiv( RCsub(1., Cexp (RCmul (-tau,
```

```
    D2))) , RCmul(gamma1*gamma1 , RCsub (1., Cmul (G2, Cexp (RC
    mul (-tau, D2)))))) , Csub (RCsub (k, RCmul (gamma1*rho12,
    Cmul(u,CI))), D2));
}
void static funcC(double t, double speed, double variance,
    double *result)
{
  if (speed==0.0 || variance==0.0)  *result=0.0;
  else  *result= variance*variance*(1.-exp(-speed*t))/(4.
    *speed);
}
void static funcLambda(double t,double speed, double initia
    l, double variance, double *result)
{
  if (speed==0.0 || variance==0.0)  *result=0.0;
  else  *result=4.*speed*initial*exp(-speed*t)/(variance*
    variance*(1.-exp(-speed*t)));
}
void static funcD(double speed, double mean, double
    variance, double *result)
{
  if (speed==0.0 || variance==0.0)  *result=0.0;
  else  *result=4.*speed*mean/(variance*variance);
}
void static Lambda1(double t, double C, double Lambda,
    double D, double *result)
{
  if (D==0.0 || Lambda==0.0)  *result=0.0;
  else  *result=sqrt(C*(Lambda-1.)+C*D+C*D/(2.*(D+Lambda)
    ));
}
void static contA(double mean, double speed, double
    variance, double *result)
{
  if (speed==0.0 || variance ==0.0)  *result=0.0;
  else  *result=sqrt(mean-variance*variance/8./speed);
}
void static contB(double initial, double A, double *result)
{
  *result=sqrt(initial)-A;
}
```

```
void static contC(double A, double B, double Lambda1,
    double *result)
{
  if (B==0.0 || (Lambda1-A)==0.0)  *result=0.0;
  else  *result=-log((Lambda1-A)/B);
}
void static Expect(double A, double B, double C, double t,
    double *result)
{
  *result = A+B*exp(-C*t);
}
void static funcA(dcomplex u, double tau, double k, double
    sigmamean, double gamma1, double lambda, double theta,
    double eta, double rho12, double rho13, double maturity, dcompl
    ex D1, dcomplex D2, dcomplex G1, dcomplex G2, double a1,
    double b1, double c1, double a2, double b2, double c2, dcomplex
    *result)
{
  double expect1=0.0, expect2=0.0;
  dcomplex A = CZERO;
  Expect(a1, b1, c1, 0, &expect1);
  Expect(a2, b2, c2, 0, &expect2);
  if (lambda==0.0 || eta==0.0)
  {
    A = RCmul (k*sigmamean/pow(gamma1, 2.), Csub(RCmul(ta
    u, RCsub(k, Cadd(RCmul(rho12*gamma1, Cmul(u, CI)), D2))),
    RCmul(2.0, Clog(Cdiv(RCsub(1., Cmul(G2, Cexp(RCmul(-tau, D2
    )))), RCsub(1., G2))))));
  }
  else
  {
    A = RCmul (k*sigmamean/pow(gamma1, 2.), Csub(RCmul(ta
    u, RCsub(k, Cadd(RCmul(rho12*gamma1, Cmul(u, CI)), D2))),
    RCmul(2.0, Clog(Cdiv(RCsub(1., Cmul(G2, Cexp(RCmul(-tau, D2
    )))), RCsub(1., G2))))));
      A = Cadd(A, RCmul (lambda*theta/pow(eta, 2.), Csub(
    RCmul(tau, RCsub(lambda, D1)), RCmul(2.0, Clog(Cdiv(RCsub(1
    ., Cmul(G1, Cexp(RCmul(-tau, D1)))), RCsub(1., G1)))))));
    A = Cadd(A, Cmul (RCmul(rho13*eta*(expect1)*(expect2)
    , Cmul(CI, u)), Csub(RCmul(tau, RCsub(lambda, D1)), RCmul(
    2.0, Clog(Cdiv(RCsub(1., Cmul(G1, Cexp(RCmul(-tau, D1)))),
```

```
     RCsub(1., G1)))))));
  }
  *result=A;
}
void static func_dc(double t, double variance, double speed
    , double *result)
{
  if (speed==0.0 || variance==0.0)  *result=0.0;
  else  *result= pow(variance, 2.)*exp(-speed*t)/4.;
}
void static func_delambda(double t, double initial, double
    speed, double variance, double *result)
{
  if (speed==0.0 || variance==0.0)  *result=0.0;
  else  *result=-4.* initial* pow (speed, 2.)* exp(speed*
    t) /(pow( exp(speed*t) -  1. , 2.) * pow (variance, 2.));
}
void static func_miu(double t, double initial, double c,
    double d, double lambda, double dc, double dlambda, double *res
    ult)
{
  if (t==0.0)  *result= sqrt(initial);
  else if(d+lambda==0.0 || c==0.0)
  {
    *result= 0.0;
  }
  else
  {
    *result= (c * (2. - d* pow((d+lambda), -2.)) * dlambd
    a + ( -2. + 2.*lambda + d* (2. + pow((d+lambda), -1.))) *
    dc)/ (2.* sqrt(2.) * sqrt( fabs (c * (-2. +2. *lambda + d*
    (2.+ 1. / (d+lambda))))));
  }
}
void static func_psi(double t, double c, double d, double
    lambda, double dc, double dlambda, double *result)
{
  if (t==0.0 )  *result =0.0;
  else if (d+lambda==0.0)
  {
    *result=0.0;
```

```
  }
  else
  {
    *result = 1./sqrt(2.) * sqrt(fabs (((d+lambda) * (d+2
    .*lambda) * dc + d* c* dlambda) / pow((d+lambda), 2.)));
  }
}

//Stochastic approximation
void static ode_bound(dcomplex u, PnlVectComplex *ybound,
    PnlVectComplex *dybound)
{
  pnl_vect_complex_set(ybound,0, Cmul(u, CI));
  pnl_vect_complex_set(dybound,1, RCadd(-1., Cmul(u, CI)))
    ;
  pnl_vect_complex_set(dybound,5, Cmul(RCmul(0.5, Cmul(u,
    CI)), CRsub(Cmul(u, CI), 1.)));
}
void static derivs(double tau, double maturity, double lam
    bda, double theta, double eta, double k, double sigmamean,
    double gamma1, double r0, double sigma0, double rho12, double rh
    o13, PnlVectComplex *y, PnlVectComplex *dy)
{
  double c_v, c_r, d_v, d_r, lambda_v, lambda_r, dc_v, dc_
    r, dlambda_v, dlambda_r, miu_v, miu_r, psi_v, psi_r;
  funcC(maturity-tau, k, gamma1, &c_v);
  funcC(maturity-tau, lambda, eta, &c_r);
  funcLambda(maturity-tau, k, sigma0, gamma1, &lambda_v);
  funcLambda(maturity-tau, lambda, r0, eta, &lambda_r);
  funcD(k, sigmamean, gamma1, &d_v);
  funcD(lambda, theta, eta, &d_r);
  func_dc(maturity-tau, gamma1, k, &dc_v);
  func_dc(maturity-tau, eta, lambda, &dc_r);
  func_delambda(maturity-tau, sigma0, k, gamma1, &dlambda_
    v);
  func_delambda(maturity-tau, r0, lambda, eta, &dlambda_r)
    ;
  func_miu(maturity-tau, sigma0, c_v, d_v, lambda_v, dc_v,
     dlambda_v, &miu_v);
  func_miu(maturity-tau, r0, c_r, d_r, lambda_r, dc_r, dl
    ambda_r, &miu_r);
```

```
func_psi(maturity-tau, c_v, d_v, lambda_v, dc_v, dlambd
  a_v, &psi_v);
func_psi(maturity-tau, c_r, d_r, lambda_r, dc_r, dlambd
  a_r, &psi_r);
  pnl_vect_complex_set(dy,1, Cadd(Cadd(Cadd(RCadd(-1.,
  pnl_vect_complex_get(y,0)), RCmul(-lambda, pnl_vect_
  complex_get(y,1))), RCmul(eta*eta/2., Csqrt(pnl_vect_complex_get(
  y,1)))), RCmul(psi_v*psi_v/2., Csqrt(pnl_vect_complex_get(
  y,3)))));
pnl_vect_complex_set(dy,2, Cadd(Cadd(RCmul(miu_v, pnl_
  vect_complex_get(y,3)), RCmul(psi_r*eta, Cmul(pnl_vect_
  complex_get(y,1), pnl_vect_complex_get(y,2)))), RCmul(psi_v*psi_v/
  2., Cmul(pnl_vect_complex_get(y,5), pnl_vect_complex_get(
  y,3)))));
pnl_vect_complex_set(dy,3, Cadd( Cadd( RCmul( eta*rho13,
   Cmul( pnl_vect_complex_get(y, 0), pnl_vect_complex_get(y,
  1))), RCmul ( rho12*psi_v, Cmul( pnl_vect_complex_get(y, 0
  ), pnl_vect_complex_get(y, 3)))), Cadd( RCmul( gamma1*psi_
  v, Cmul( pnl_vect_complex_get(y, 4), pnl_vect_complex_get(
  y, 3))), RCmul( eta*psi_r, Cmul( pnl_vect_complex_get(y, 1)
  , pnl_vect_complex_get(y, 3))))));
pnl_vect_complex_set(dy,4, Cadd (Cadd( Csub( RCmul(1./2.
  , Cmul( pnl_vect_complex_get(y, 0), CRsub( pnl_vect_
  complex_get(y, 0), 1.0))), RCmul( k, pnl_vect_complex_get(y, 4))),
   Cadd( RCmul( gamma1*rho12, Cmul(pnl_vect_complex_get(y, 0
  ), pnl_vect_complex_get(y, 4))), RCmul( gamma1*gamma1/2.,
  Csqrt(pnl_vect_complex_get(y, 4))))), Cadd ( RCmul( rho13*
  psi_r, Cmul( pnl_vect_complex_get(y, 0), pnl_vect_complex_g
  et(y, 3))), RCmul( psi_v*psi_v/2., Csqrt(pnl_vect_complex_g
  et(y, 3))))));
pnl_vect_complex_set(dy,5, Cadd( Cadd( Cadd( RCmul( miu_
  r, pnl_vect_complex_get(y, 3)), RCmul( psi_v*rho12, Cmul(
  pnl_vect_complex_get(y, 0), pnl_vect_complex_get(y, 5)))),
  Cadd( RCmul( gamma1*psi_v, Cmul( pnl_vect_complex_get(y,4),
  pnl_vect_complex_get(y,5))), RCmul( rho13*psi_r, Cmul( pnl_
  vect_complex_get(y, 0), pnl_vect_complex_get(y, 2))))), RCmu
  l( psi_r*psi_v, Cmul( pnl_vect_complex_get(y, 2), pnl_vect_
  complex_get(y, 3)))));
pnl_vect_complex_set(dy,6, Cadd(Cadd( Cadd( RCmul( k*si
  gmamean, pnl_vect_complex_get(y, 4)), RCmul( lambda*theta,
  pnl_vect_complex_get(y, 1))), Cadd( RCmul( miu_v, pnl_vect_
```

```
    complex_get(y, 5)), RCmul( miu_r, pnl_vect_complex_get(y, 2)))),
    Cadd( RCmul( psi_v*psi_v/2., Csqrt(pnl_vect_complex_get(y, 5
    ))), RCmul( psi_r*psi_v/2., Csqrt(pnl_vect_complex_get(y,
    2))))));
}
void static ode_solution_step(dcomplex u, double maturity,
    double lambda, double theta, double eta, double r0, double k,
    double sigmamean, double gamma1, double sigma0, double rho12,
    double rho13, double bound, double step, int n, PnlVectComplex *
    ybound, PnlVectComplex *dybound, PnlVectComplex *yout)
{

  int i;
  double xh, hh, h6;
  PnlVectComplex *dym, *dyt, *yt;
  dym=pnl_vect_complex_create_from_dcomplex(n, Complex(0.0
    , 0.0));
  dyt=pnl_vect_complex_create_from_dcomplex(n, Complex(0.0
    , 0.0));
  yt=pnl_vect_complex_create_from_dcomplex(n, Complex(0.0,
     0.0));
  hh=step*0.5;
  h6=step/6.0;
  xh=bound+hh;
  for (i=0; i<n; i++)  pnl_vect_complex_set(yt, i, Cadd(
    pnl_vect_complex_get(ybound, i), RCmul( hh, pnl_vect_
    complex_get(dybound, i))));
  derivs(xh, maturity, lambda, theta, eta, k, sigmamean,
    gamma1, r0, sigma0, rho12, rho13, yt, dyt);
  for (i=0; i<n; i++)  pnl_vect_complex_set(yt, i, Cadd(
    pnl_vect_complex_get(ybound, i), RCmul( hh, pnl_vect_
    complex_get(dyt, i))));
  derivs(xh, maturity, lambda, theta, eta, k, sigmamean,
    gamma1, r0, sigma0, rho12, rho13, yt, dym);
  for (i=0; i<n; i++)
  {
    pnl_vect_complex_set(yt, i, Cadd( pnl_vect_complex_g
    et(ybound, i), RCmul( step, pnl_vect_complex_get(dym, i))))
    ;
    pnl_vect_complex_set(dym, i, Cadd( pnl_vect_complex_g
    et(dym, i), pnl_vect_complex_get(dyt, i)));
```

```
  }
  derivs(bound+step, maturity, lambda, theta, eta, k, si
    gmamean, gamma1, r0, sigma0, rho12, rho13, yt, dyt);
  for (i=0; i<n; i++)  pnl_vect_complex_set(yout, i, Cad
    d( pnl_vect_complex_get(ybound, i), RCmul( h6, Cadd (Cadd(
    pnl_vect_complex_get(dybound, i), pnl_vect_complex_get(dyt,
     i)), RCmul( 2.0, pnl_vect_complex_get(dym, i))) )));
  pnl_vect_complex_free(&dym);
  pnl_vect_complex_free(&dyt);
  pnl_vect_complex_free(&yt);
}
void static ode_solution(dcomplex u, double maturity,
    double lambda, double theta, double eta, double r0, double k,
    double sigmamean, double gamma1, double sigma0, double rho12,
    double rho13, double x1, double x2, int nvar, int nstep, PnlVec
    tComplex *ybound, PnlVectComplex *vout)
{
  int i, j;
  double x, h;
  PnlVectComplex *v, *dv;
  v=pnl_vect_complex_create_from_dcomplex(nvar, Complex(0.
    0, 0.0));
  dv=pnl_vect_complex_create_from_dcomplex(nvar, Complex(0
    .0, 0.0));
  for (i=0; i<nvar; i++)  pnl_vect_complex_set(v, i, pnl_
    vect_complex_get(ybound, i));
  x=x1;
  h=(x2-x1)/nstep;
  for (j=1; j<=nstep; j++)
  {
    derivs(x, maturity, lambda, theta, eta, k, sigmamean,
     gamma1, r0, sigma0, rho12, rho13, v, dv);
    ode_solution_step(u, maturity, lambda, theta, eta, r0
    , k, sigmamean, gamma1, sigma0, rho12, rho13, x, h, nvar,
    v, dv, vout);
     if ((double)(x+h)==x)  printf("Step size too small
    in routine");
     x += h;
     for (i=0; i<nvar; i++)  pnl_vect_complex_set(v, i,
    pnl_vect_complex_get(vout, i));
  }
```

```
  pnl_vect_complex_free(&v);
  pnl_vect_complex_free(&dv);
}


void static chf(dcomplex u, double kappa, double sigmamean,
     double gamma1, double sigma0, double lambda, double thet
    a, double eta, double r0, double rho12, double rho13,
    double S0, double K, double maturity, double a, int approximatio
    n_flag, dcomplex *result)
{

  dcomplex expon = Complex(0.0, 0.0);
  if (approximation_flag==0)
  {
  double a1, a2, b1, b2, c1, c2, fc1, fc2, fd1, fd2, flamb
    da1, flambda2, clambda1, clambda2;
  dcomplex D1, D2, G1, G2, A, Bx, Bsigma, Br;
  double tau;
  tau=maturity-0.0;
  funcD1(u, lambda, eta, &D1);
  funcD2(u, gamma1, rho12, kappa, &D2);
  funcG1(lambda, D1, &G1);
  funcG2(u, kappa, gamma1, rho12, D2, &G2);
  funcC(1.0, kappa, gamma1, &fc1);
  funcC(1.0, lambda, eta, &fc2);
  funcLambda(1.0, kappa, sigma0, gamma1, &flambda1);
  funcLambda(1.0, lambda, r0, eta, &flambda2);
  funcD(kappa, sigmamean, gamma1, &fd1);
  funcD(lambda, theta, eta, &fd2);
  Lambda1(1.0, fc1, flambda1, fd1, &clambda1);
  Lambda1(1.0, fc2, flambda2, fd2, &clambda2);
  contA(sigmamean, kappa, gamma1, &a1);
  contA(theta, lambda, eta, &a2);
  contB(sigma0, a1, &b1);
  contB(r0, a2, &b2);
  contC(a1, b1, clambda1, &c1);
  contC(a2, b2, clambda2, &c2);
  funcBx(u, &Bx);
  funcBr(u, tau, lambda, eta, D1, G1, &Br);
  funcBsigma(u, tau, kappa, gamma1, rho12,  D2, G2, &Bsi
```

```
    gma);
  funcA(u, tau, kappa, sigmamean, gamma1, lambda, theta,
    eta, rho12, rho13, maturity, D1, D2, G1, G2, a1, b1, c1, a2
    , b2, c2, &A);

   expon = Cadd (A, RCmul(sigma0, Bsigma));
     expon = Cadd (expon, RCmul(r0, Br));
  *result= Cexp(expon);
  }
  else
  {
    PnlVectComplex *ybound, *dybound, *yout, *vout;
    ybound=pnl_vect_complex_create_from_dcomplex(7,
    Complex(0.0,0.0));
    dybound=pnl_vect_complex_create_from_dcomplex(7,
    Complex(0.0,0.0));
    yout=pnl_vect_complex_create_from_dcomplex(7,Complex(
    0.0,0.0));
    vout=pnl_vect_complex_create_from_dcomplex(7,Complex(
    0.0,0.0));
    ode_bound(u, ybound, dybound);
    ode_solution(u, maturity, lambda, theta, eta, r0, ka
    ppa, sigmamean, gamma1, sigma0, rho12, rho13, 0.0, maturit
    y, 7, 3, ybound, vout);

    expon = pnl_vect_complex_get(vout, 6);
    expon = Cadd(expon, RCmul(r0, pnl_vect_complex_get(
    vout, 1)));
    expon = Cadd(expon, RCmul(sigma0, pnl_vect_complex_g
    et(vout, 4)));
    *result=Cexp(expon);
    pnl_vect_complex_free(&ybound);
       pnl_vect_complex_free(&dybound);
    pnl_vect_complex_free(&yout);
    pnl_vect_complex_free(&vout);
  }
}

//COSINE method to calculate inverse fourier integral
static  double cosine_function_xi(double d,double c,double
    dma,double cma, double fnpi)
```

```
{


    double res= 1./(1+fnpi*fnpi)*( ( cos(dma) + fnpi *sin(
    dma)) * exp(d) - ( cos(cma) + fnpi *sin(cma)) *exp(c));
  return res;
}

static double cosine_function_psi(double d, double c,
    double dma, double cma, double fnpi)
{

    double res=(fnpi==0.)?(d-c): (sin(dma)-sin(cma))/fnpi;
    return res;
}

static double cosine_Vk_call(double K,double a,double b,
    double fnpi)
{
  double ma=-a*fnpi;
  double bma=(b-a)*fnpi;
  return 2. /(b-a) * K *(cosine_function_xi(b,0,bma,ma,fnpi
    )-cosine_function_psi(b,0,bma,ma,fnpi));
}
static double cosine_Vk_put(double K,double a,double b,
    double fnpi)
{

  double ma=-a*fnpi;
  return 2. /(b-a) * K *(-cosine_function_xi(0,a,ma,0,fnpi)
    +cosine_function_psi(0,a,ma,0,fnpi));
}
static double cosine_Vk(int callput_flag, double K, double
    a, double b, double fnpi)
{

  double result=0.0;
  if (callput_flag==0)  result=cosine_Vk_call(K, a, b, fn
    pi);
  else  result=cosine_Vk_put(K, a, b, fnpi);
  return result;
```

```
}
static void cosine_left_bound(double L, double S0, double
    K, double sigmamean, double sigma0, double eta, double gam
    ma1, double kappa, double r0, double tau, double rho,
    double *a, double *b)
{

  double c1=0.0, c2=0.0;
  c1=r0*tau+(1.-exp(-kappa*tau))*(sigmamean-sigma0)/(2.0*
    kappa)-0.5*sigmamean*tau;
  c2=(1.0/(8.0*pow(kappa,3)))*(gamma1*tau*kappa*exp(-kapp
    a*tau)*(sigma0-sigmamean)*(8.0*kappa*rho-4.0*gamma1)+kappa*
    rho*gamma1*(1.0-exp(-kappa*tau))*(16.0*sigmamean-8.0*sigma0
    )+2.0*sigmamean*kappa*tau*(-4.0*kappa*rho*gamma1+pow(gam
    ma1,2)+4*pow(kappa,2))+pow(gamma1,2)*((sigmamean-2.0*sigma0)
    *exp(-2*kappa*tau)+sigmamean*(6.0*exp(-kappa*tau)-7)+2*si
    gma0)+8*pow(kappa,2)*(sigma0-sigmamean)*(1-exp(-kappa*tau)))
    ;

  /*Truncation range*/
  *a=c1-L*pow(fabs(c2),0.5)+log(S0/K);
  *b=c1+L*pow(fabs(c2),0.5)+log(S0/K);
}

static void HCIR_COSINE(double kappa, double sigmamean,
    double gamma1, double sigma0, double lambda, double theta,
    double eta, double r0, double rho12, double rho13, double K,
    double S0, double maturity, double L, int N, int callput_flag,
    int approximation_flag, double *Price, double *GreekDelta)
{

  double sum=0, sumd=0, a=0, b=0;
  double x = log(S0/K);
  double invbma=0.0;
  double xma=0.0;
    dcomplex fnpi=CZERO;
    dcomplex phi=CZERO;
  int i;
  cosine_left_bound(L,S0, K,sigmamean,sigma0, eta,gamma1,
    kappa,r0, maturity, rho12, &a, &b);
  xma=x-a;
```

```
  invbma=M_PI/(b-a);
  chf(fnpi, kappa, sigmamean, gamma1, sigma0, lambda, thet
    a, eta, r0, rho12, rho13, S0, K, maturity, a, approximatio
    n_flag, & phi);
  sum=0.5*phi.r*cosine_Vk(callput_flag,K,a,b,fnpi.r);
  sumd=0.;
  for(i=1; i<N; i++)
        {
          fnpi.r+=invbma;
    chf(fnpi, kappa, sigmamean, gamma1, sigma0, lambda,
    theta, eta, r0, rho12, rho13, S0, K, maturity, a, approxima
    tion_flag, & phi);
          sum +=(phi.r*cos(fnpi.r*xma)-phi.i*sin(fnpi.r*x
    ma))*cosine_Vk(callput_flag,K,a,b,fnpi.r);
          sumd-=(phi.i*cos(fnpi.r*xma)+phi.r*sin(fnpi.r*x
    ma))*fnpi.r*cosine_Vk(callput_flag,K,a,b,fnpi.r);
        }
    *GreekDelta=sumd/S0;
    *Price=sum;
}

int ApGrzelakOosterlee(double spot,NumFunc_1  *p, double
    maturity,double r0, double kr,double rbar, double sigmar,
    double v0,double kv,double vbar,double sigmav,double rho12,
    double rho13,int approximation_flag,double *ptprice,double *ptde
    lta)
{
  double K,price,delta;
  int N,L;
  int flag_call;


  //ATTENTION Correlation rv inthis algorithm.

  N=100; // discrect steps
  L=10; //parameter for integration bound

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

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

```
  else
    flag_call=1;

   HCIR_COSINE(kv, vbar, sigmav,v0, kr, rbar, sigmar, r0,
    rho12, rho13,K, spot, maturity, L, N,flag_call, approxima
    tion_flag, &price, &delta);

  //Price
   *ptprice=price;


  //Delta
  *ptdelta=delta;

  return OK;
}

int CALC(AP_GrzelakOosterlee)(void *Opt, void *Mod, Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

return ApGrzelakOosterlee(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_PDOUBLE,
                          ptMod->RhoSV.Val.V_PDOUBLE,
                          Met->Par[0].Val.V_ENUM.value,
                          &(Met->Res[0].Val.V_DOUBLE),
                          &(Met->Res[1].Val.V_DOUBLE));
}
```

```
static int CHK_OPT(AP_GrzelakOosterlee)(void *Opt, void *
    Mod)
{
  if ((strcmp( ((Option*)Opt)->Name,"PutEuro")==0)||(strcmp
    ( ((Option*)Opt)->Name,"CallEuro")==0))
    return OK;
  return WRONG;
}
#endif //PremiaCurrentVersion
static PremiaEnumMember ApproximationMembers[] =
{
  { "Deterministic approximation", 0 },
  { "Stochastic approximation", 1 },
  { NULL, NULLINT }
};

static DEFINE_ENUM(Approximation,ApproximationMembers);
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;
      Met->HelpFilenameHint = "ap_grzelak_oosterlee";
      Met->Par[0].Val.V_ENUM.value=0;
      Met->Par[0].Val.V_ENUM.members=&Approximation;
    }

  return OK;
}

PricingMethod MET(AP_GrzelakOosterlee)=
{
  "AP_GrzelakOosterlee",
  { {"Approximation method (Rho r V=0)",ENUM,{100},ALLOW},
    {" ",PREMIA_NULLTYPE,{0},FORBID}
  },
  CALC(AP_GrzelakOosterlee),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
```

```
  CHK_OPT(AP_GrzelakOosterlee),
  CHK_ok,
  MET(Init)
};
```

References