Help

```
#include  "heshw1d_std.h"
#include "enums.h"
#include "error_msg.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_matrix.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2015+2) //The "#els
static int CHK_OPT(AP_FOURIERCOSINE_HESHW1D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_FOURIERCOSINE_HESHW1D)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


/*ZCB Data*/
static double *initial_yield; /*Values of initial_yield*/
static double *initial_forward; /*Values of initial_forward*/
static double *initial_derive_forward; /*Values of initial_derive_forward*/
static double *t_vect; /*Times used in the temporal loop of the scheme.*/
static double *tm; /*Times T of maturities read in the file initialyield.dat */
static double *Pm; /*Values of the zero coupon P(0,tm) read in the file initialy
static char *init_tr;

static int lecture_tr()
{
  int i;
  char ligne[30];
  char *pligne;
  double p, tt_value;
  FILE *Entrees;

  Entrees = fopen(init_tr, "r");
```

```
  if (Entrees == NULL)
    {
  printf("LE FICHIER N'A PU ETRE OUVERT. VERIFIEZ LE CHEMIN.\ n");
    }

  /* i is the number of lines that has been read */
  i = 0;
  pligne = ligne;
  Pm = (double *)malloc(200 * sizeof(double));
  tm = (double *)malloc(200 * sizeof(double));

  while (1)
    {
  pligne = fgets(ligne, sizeof(ligne), Entrees);
  if (pligne == NULL) break;
  else
        {
  sscanf(ligne, "%lf t=%lf", &p, &tt_value);
  /* The line read must be written "0.943290 t=0.5" where 0.943290 is a double f
  Pm[i] = p; /*save the price of the zero coupon*/
  tm[i] = tt_value; /*save the corresponding time*/
  i++;

        }
    }
  fclose(Entrees);

  return i;
}



static void compute_forward(double *Price, double *Time, int size, double *Forwa
{
  //Forward = - d log(P) / dt
  //Derive_Forward = d Forward / dt
  int i;

  for (i=0; i<=size-1; i++)
    {
```

```
  Forward[i] = -(log(Price[i+1])-log(Price[i]))/(Time[i+1]-Time[i]);
    }
  Forward[size] = Forward[size-1];

  for (i=0; i<=size-1; i++)
    {
  Derive_Forward[i] = (Forward[i+1]-Forward[i])/(Time[i+1]-Time[i]);
    }
  Derive_Forward[size] = Derive_Forward[size-1];
}


static void interpole(int n_price, int imax, double *t)
{
  int i, iF, j;

  n_price--;

  i = 0;
  while (t[i] <= tm[1])
    {
  initial_yield[i] = (tm[1] - t[i]) / tm[1] + t[i] * Pm[1] / tm[1];
  i++;
    }

  for (j = 0; j < n_price; j++)
    {
  while (i <= imax && t[i] < tm[j + 1] )
        {
  initial_yield[i] = (tm[j + 1] - t[i]) / (tm[j + 1] - tm[j]) * Pm[j] + (t[i] -
  i++;
        }
    }

  if (i <= imax  && t[i] > tm[n_price])
    {
  for (iF = i ; iF <= imax ; iF++)
        {
  initial_yield[iF] = Pm[n_price] + (Pm[n_price] - Pm[n_price - 1]) / (tm[n_pric
        }
    }
```

```
}


static void extract_forward(double *Forward, double *Derive_Forward, double *Tim
{
  int i;
  int j;

  i=0;
  j=0;

  while (j<=size_extract)
    {
  while ((i<=size)&&(Time[i]<t_extract[j]) )
i++;

  initial_forward[j] = Forward[i];
  initial_derive_forward[j] = Derive_Forward[i];

  j++;
    }
}

/* static void compute_forward(double *Price, double *Time, int size, double *Fo
/* { */
/*   //Forward = - d log(P) / dt */
/*   //Derive_Forward = d Forward / dt */
/*   int i; */

/*   for (i=0; i<=size-1; i++) */
/*     { */
/*    Forward[i] = -(log(Price[i+1])-log(Price[i]))/(Time[i+1]-Time[i]); */
/*     } */
/*   Forward[size] = Forward[size-1]; */

/*   for (i=0; i<=size-1; i++) */
/*     { */
/*    Derive_Forward[i] = (Forward[i+1]-Forward[i])/(Time[i+1]-Time[i]); */
/*     } */
/*   Derive_Forward[size] = Derive_Forward[size-1]; */
/* } */
```

```
/* static void interpole(int n_price, int imax, double *t) */
/* { */
/*   int i, iF, j; */

/*   n_price--; */

/*   i = 0; */
/*   while (t[i] <= tm[1]) */
/*     { */
/*    initial_yield[i] = (tm[1] - t[i]) / tm[1] + t[i] * Pm[1] / tm[1]; */
/*    i++; */
/*     } */

/*   for (j = 0; j < n_price; j++) */
/*     { */
/*    while (t[i] < tm[j + 1] && i <= imax + 1) */
/*         { */
/*    initial_yield[i] = (tm[j + 1] - t[i]) / (tm[j + 1] - tm[j]) * Pm[j] + (t[i
/*    i++; */
/*         } */
/*     } */

/*   if (t[i] > tm[n_price] && i <= imax + 1) */
/*     { */
/*    for (iF = i ; iF <= imax ; iF++) */
/*         { */
/*    initial_yield[iF] = Pm[n_price] + (Pm[n_price] - Pm[n_price - 1]) / (tm[n_
/*         } */
/*     } */
/* } */


/* static void extract_forward(double *Forward, double *Derive_Forward, double *
/* { */
/*   int i; */
/*   int j; */

/*   i=0; */
/*   j=0; */

/*   while (j<size_extract) */
```

```
/*     { */
/*    while ((Time[i]<t_extract[j]) && (i<size)) */
/*  i++; */

/*    initial_forward[j] = Forward[i]; */
/*    initial_derive_forward[j] = Derive_Forward[i]; */

/*    j++; */
/*     } */
/* } */

/*  Fourier Cosine Method */

void static funcD1(dcomplex u, double kv, double sigmav, double rho12, dcomplex
{
  if (kv==0.0 || sigmav==0.0) *result=CZERO;
  else *result=Csqrt(Csub(Cpow_real(CRsub(RCmul(sigmav*rho12, Cmul(CI,u)), kv),
}
void static funcg(dcomplex u, double kv, double sigmav, double rho12, dcomplex D
{
  if (Cimag(D1)==0.0 && Creal(D1)==0.0) *result=CZERO;
  else *result=Cdiv(Csub(RCsub(kv, RCmul(sigmav*rho12, Cmul(CI,u))), D1), Cadd(R
}
void static funcBx(dcomplex u, dcomplex *result)
{
  *result= Cmul (u, CI);
}
void static funcBr(dcomplex u, double tau, double kappar, double sigmar, dcomple
{
  if(sigmar==0.0)
{
  if (kappar==0.0) *result=RCmul(tau, CRsub(Cmul(u, CI), 1.0));
  else *result = RCmul((1.0-exp(-kappar*tau))/kappar, CRsub(Cmul(u, CI), 1.0));
}
  else *result= RCmul((1.- exp(-tau*kappar))/kappar,  CRsub(Cmul(u, CI), 1.0));
}
void static funcBv(dcomplex u, double tau, double kappav, double sigmav, double
{
  if (sigmav==0.0)
{
  if (kappav==0.0) *result = RCmul(-tau/2.0, Cadd(Cmul(u, u), Cmul(u, CI)));
```

```
  else *result = RCmul((exp(-kappav*tau)-1.0)/(2.0*kappav), Cadd(Cmul(u, u), Cmu
}
  else *result=Cmul (Cdiv( RCsub(1., Cexp (RCmul (-tau, D1))) , RCmul(sigmav*sig
}
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 initial, double variance, d
{
  if (speed==0.0 || variance==0.0)        *result=0.0;
  else    *result=4.*speed*initial*exp(-speed*t)/(variance*variance*(1.-exp(-spe
}
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 funcI1(dcomplex u, double tau, double kr, double sigmar, double r0,
{
  if (flat_flag==0)
```

```
    {
  *result = CRmul(CRsub(Cmul(u, CI), 1.0), kr*r0*tau + r0* (exp(-kr*tau)-1.) + p
    }
  else if (flat_flag==1)
    {
  *result= CRmul(CRsub(Cmul(u, CI), 1.0), r0*(exp(-kr*tau)-1.)+ kr*(log(ZCB1)-lo
    }
}
void static funcI2 (dcomplex u, double tau, double kv, double sigmav, double rho
{
  *result = Csub( RCmul( tau/(sigmav*sigmav), RCsub(kv, Cadd(RCmul(sigmav*rho12,
}
void static funcI3 (dcomplex u, double tau, double kr, dcomplex *result)
{
  *result = RCmul(1./(2.*pow(kr, 3.))*(3.+exp(-2.*kr*tau)-4.*exp(-kr*tau)-2*kr*t
}
void static funcI4 (dcomplex u, double tau, double kr, double a, double b, doubl
{
  *result = RCmul(-1./kr*(b/c*(1.0-exp(-c*tau))+a*tau+a/kr*(exp(-kr*tau)-1.)+b/(
}
void static funcA(dcomplex u, double tau, double kr, double thetar, double sigma
{
  *result=Csub(Cadd(Cadd(RCmul(1., I1), RCmul(kv*thetav, I2)), Cadd(RCmul(sigmar
}

void static chf(dcomplex u, double kappav, double thetav, double sigmav, double
{
  double a, b, c, fc1, fd1, flambda1, clambda;
  dcomplex expon, D1, g, I1, I2, I3, I4, A, Bx, Bv, Br;
  double tau;

  expon = Complex(0.0, 0.0);
  tau=maturity-0.0;
  funcD1(u, kappav, sigmav, rho12, &D1);
  funcg(u, kappav, sigmav, rho12, D1, &g);
  funcC(1.0, kappav, sigmav, &fc1);
  funcLambda(1.0, kappav, v0, sigmav, &flambda1);
  funcD(kappav, thetav, sigmav, &fd1);
  Lambda1(1.0, fc1, flambda1, fd1, &clambda);
  contA(thetav, kappav, sigmav, &a);
  contB(v0, a, &b);
```

```
  contC(a, b, clambda, &c);
  funcI1(u, tau, kappar, sigmar, r0, ZCB1, ZCB2, flat_flag, &I1);
  funcI2(u, tau, kappav, sigmav, rho12, g,D1, &I2);
  funcI3 (u, tau, kappar, &I3);
  funcI4 (u, tau, kappar, a, b, c, &I4);
  funcBx(u, &Bx);
  funcBr(u, tau, kappar, sigmar, D1, g, &Br);
  funcBv(u, tau, kappav, sigmav, rho12,  D1, g, &Bv);
  funcA(u, tau, kappar, thetar, sigmar, kappav, thetav, sigmav, rho13, divident,
  expon = Cadd (A, RCmul(v0, Bv));
  expon = Cadd (expon, RCmul(r0, Br));
  expon = Cadd (expon, RCmul(log(S0/K), Bx));
  *result= Cexp(expon);

}

//COSINE method to calculate inverse fourier integral
static double cosine_function_xi(double d,double c,double dma,double cma, double
{
  double res;
  res= 1./(1+fnpi*fnpi)*( ( cos(dma) + fnpi *sin(dma)) * exp(d) - ( cos(cma) + f
  return res;
}
static double cosine_function_psi(double d, double c, double dma, double cma, do
{
  double res;
  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, bma;
  ma=-a*fnpi;
  bma=(b-a)*fnpi;
  return 2. /(b-a) * K *(cosine_function_xi(b,0,bma,ma,fnpi)-cosine_function_psi
}
static double cosine_Vk_put(double K,double a,double b, double fnpi)
{
  double ma;
  ma=-a*fnpi;
```

```
  return 2. /(b-a) * K *(-cosine_function_xi(0,a,ma,0,fnpi)+cosine_function_psi(
}
static double cosine_Vk(int callput_flag, double K, double a, double b, double f
{
  double result;
  result=0.0;
  if (callput_flag==1) result=cosine_Vk_call(K, a, b, fnpi);
  else result=cosine_Vk_put(K, a, b, fnpi);
  return result;
}
void static cosine_left_bound(double L, double S0, double K, double sigmamean, d
{

  double c1, c2;
  c1=0.0;
  c2=0.0;
  c1=r0*tau+(1.-exp(-kappa*tau))*(sigmamean-sigma0)/(2.0*kappa)-0.5*sigmamean*ta
  c2=(1.0/(8.0*pow(kappa,3)))*(gamma1*tau*kappa*exp(-kappa*tau)*(sigma0-sigmamea

  /*Truncation range*/
  *a=c1-L*pow(fabs(c2),0.5)+log(S0/K);
  *b=c1+L*pow(fabs(c2),0.5)+log(S0/K);
}
/*Compute Price Option*/
int  AP_FourierCosine_HesHW(int callput_flag, double S0, NumFunc_1  *p, double m

{
  double ZCB1=1.,ZCB2;
  double ZCB_maturity;
  int NinterpolZCB;
  int Nt=100;
  int n_price;
  double *t_interpol;
  double *forward;
  double *derive_forward;
  double sum, sumd, a, b, vk, x, invbma, xma;
  dcomplex fnpi, phi;
  int i;
  double K;
  double L,rho12,rho13;
```

```
  if ((fabs(rhorv) > 0))
    return UNTREATED_CASE;

  // Compute the mean reversion from Zero Coupon Bond data curve.
  // Calibration on the zero coupon bond data table.
  // Read the values of pm and tm.
  if(flat_flag==1)
{
  init_tr = curve;
  n_price = lecture_tr();
  // We search in initialyield.dat the biggest value before time T.
  if (maturity > tm[n_price - 1])
{
  printf("\ nError : time bigger than the last time value entered in %s\ n\ n",
}
  for(i=0; i<n_price; i++)
{
  if(tm[i]==maturity)
{
  ZCB_maturity=Pm[i];
  //printf("n_price =%d, ZCB_maturity=%f\ n", n_price, ZCB_maturity);
}
}
  ZCB2=ZCB_maturity;

  // Compute interpolation on a very fine grid given pm, tm and n_price. Return
  NinterpolZCB = MAX(Nt,-MAX(-Nt*1000,-1000000)); // Moins que 1.000.000 mais qu
  NinterpolZCB = tm[n_price - 1] * NinterpolZCB;
  initial_yield = (double *)malloc((NinterpolZCB + 1) * sizeof(double));
  t_interpol = (double *)malloc((NinterpolZCB + 1) * sizeof(double));
  for (i = 0; i <= NinterpolZCB; i++)
{
  initial_yield[i] = 0.;
  t_interpol[i] = i * (maturity/NinterpolZCB);
}
  interpole(n_price, NinterpolZCB, t_interpol);



  // Compute the forward rate and the derivative of the forward rate given a ver
  forward = (double *)malloc((NinterpolZCB + 1) * sizeof(double));
```

```
  derive_forward = (double *)malloc((NinterpolZCB + 1) * sizeof(double));
  for (i = 0; i <= NinterpolZCB; i++)
{
  forward[i] = 0.;
  derive_forward[i] = 0.;
}

  compute_forward(initial_yield, t_interpol, NinterpolZCB, forward, derive_forwa

  // We obtain the forward and derivative of forward rate on a very fine grid.
  // We should now define the rate on the real temporal grid.
  initial_forward = (double *)malloc((Nt + 1) * sizeof(double));
  initial_derive_forward = (double *)malloc((Nt + 1) * sizeof(double));
  t_vect = (double *)malloc((Nt + 2) * sizeof(double));
  for (i = 0; i < Nt+1; i++)
{
  initial_forward[i] = 0.;
  initial_derive_forward[i] = 0.;
  t_vect[i] = i * maturity/(double)Nt;
}
  extract_forward(forward, derive_forward, t_interpol, NinterpolZCB, t_vect, Nt)
  r0=forward[0];

  free(Pm);
  free(tm);
  free(t_vect);
  free(initial_derive_forward);
  free(initial_forward);
  free(initial_yield);
  free(forward);
  free(derive_forward);
  free(t_interpol);
}

  K = p->Par[0].Val.V_PDOUBLE;
  L = 10; //Integration Domain
  rho12 = rhoSv;
  rho13 = rhoSr;

  sum=0;
  sumd=0;
```

```
  a=0;
  b=0;

  x = log(S0/K);
  invbma=0.0;
  xma=0.0;
  fnpi=CZERO;
  phi=CZERO;

  cosine_left_bound(L, S0, K, thetav, v0, sigmar, sigmav, kappav, r0, maturity,

  xma=x-a;
  invbma=M_PI/(b-a);
  if(flat_flag==0)
chf(fnpi, kappav, thetav, sigmav, v0, kappar, 0., sigmar, r0, rho12, rho13, divi
  else
chf(fnpi, kappav, thetav, sigmav, v0, kappar,0., sigmar, r0, rho12, rho13, divid

  vk=cosine_Vk(callput_flag,K,a,b,fnpi.r);
  sum=0.5*phi.r*vk;
  sumd=0.;

  for(i=1; i<N; i++)
{
  fnpi.r+=invbma;
  if(flat_flag==0)
chf(fnpi, kappav, thetav, sigmav, v0, kappar, 0., sigmar, r0, rho12, rho13, divi
  else
chf(fnpi, kappav, thetav, sigmav, v0, kappar,0., sigmar, r0, rho12, rho13, divid
  vk=cosine_Vk(callput_flag,K,a,b,fnpi.r);
  sum +=(phi.r*cos(fnpi.r*xma)-phi.i*sin(fnpi.r*xma))*cosine_Vk(callput_flag,K,a
  sumd-=(phi.i*cos(fnpi.r*xma)+phi.r*sin(fnpi.r*xma))*fnpi.r*cosine_Vk(callput_f
}
  *ptdelta=sumd/S0;
  *ptprice=sum;

  return OK;
}

int CALC(AP_FOURIERCOSINE_HESHW1D)(void *Opt, void *Mod, PricingMethod *Met)
{
```

```
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double divid;
  int iscall;

  divid = ptMod->divid.Val.V_DOUBLE;
  iscall = FALSE;
  if (ptOpt->PayOff.Val.V_NUMFUNC_1->Compute == &Call) iscall = TRUE;


  return AP_FourierCosine_HesHW(iscall, ptMod->S0.Val.V_PDOUBLE,
                                ptOpt->PayOff.Val.V_NUMFUNC_1,
                                ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DAT
 ptMod->flat_flag.Val.V_INT,
                               MOD(GetYield)(ptMod),
                               MOD(GetCurve)(ptMod),
                               ptMod->kr.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,
                                ptMod->RhorV.Val.V_PDOUBLE,
                                Met->Par[0].Val.V_PINT,
&(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(AP_FOURIERCOSINE_HESHW1D)(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->init = 1;
      Met->HelpFilenameHint = "ap_fouriercosine_heshw1d";
      Met->Par[0].Val.V_INT2 = 128;
    }

  return OK;
}

PricingMethod MET(AP_FOURIERCOSINE_HESHW1D) =
{
  "AP_FourierCosine",
  {{"Number of integration steps (power of 2)", INT2, {100}, ALLOW}, {" ", PREMI
  CALC(AP_FOURIERCOSINE_HESHW1D),
  { {"Price", DOUBLE, {100}, FORBID},{"Delta", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(AP_FOURIERCOSINE_HESHW1D),
  CHK_ok,
  MET(Init)
};
```