Help

```
#include  "
href../../../mod/uvm1d/uvm1d_std/uvm1d_std_h_src.pdfuvm1d_std.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"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_root.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2014+2) //The "#els
static int CHK_OPT(AP_UVMFOURIERCOSINE)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_UVMFOURIERCOSINE)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

void static fxi(double a, double b, double l, double u, int n, double *result)
{
  *result = (n == 0) ? (exp(u) - exp(l)) : 1. / (1. + pow(n * M_PI / (b - a) , 2
}
void static fpsi(double a, double b, double l, double u, int n, double *result)
{
  *result = (n == 0) ? (u - l) : (sin(n * M_PI * (u - a) / (b - a)) - sin(n * M_
}

void static mbasic(double a, double b, double l, double u, int N, PnlVectComplex
{

  int n;
  pnl_vect_complex_set(mj, 0, RCmul((u - l)*M_PI / (b - a), CI));
  for (n = 1; n < 2 * N + 1; n++)
    {
      pnl_vect_complex_set(mj, n, CRdiv(Csub(Cexp(RCmul(n * (u - a)*M_PI / (b -
    }
```

```
}
void static fphi(int N, double deltat, double r, double divid, double sigma, dou
{
  double c1 = (r - divid - 0.5 * pow(sigma, 2)) * deltat;
  double c2 = pow(sigma, 2) * deltat;
  double omegaj;
  int j;

  for (j = 0; j < N; j++)
    {
      omegaj = j * M_PI / (b - a);
      pnl_vect_complex_set(phi, j, Cexp(CRsub(Cadd(Complex(0, omegaj * c1), Comp
    }
}

void static fbeta(int N, PnlVectComplex *phi, PnlVect *ve, PnlVectComplex *beta)
{
  int j;

  for (j = 0; j < N; j++)
    {
      pnl_vect_complex_set(beta, j, RCmul(pnl_vect_get(ve, j), pnl_vect_complex_
    }

}

void static c_fft(int domain, int number, double T, int M, double a, double b, d
{
  PnlVectComplex *ms = pnl_vect_complex_create(2 * number);
  PnlVectComplex *mc = pnl_vect_complex_create(2 * number);
  PnlVectComplex *us = pnl_vect_complex_create(2 * number);
  PnlVectComplex *ivector1 = pnl_vect_complex_create(2 * number);
  PnlVectComplex *ivector2 = pnl_vect_complex_create(2 * number);
  int n;

  for (n = 0; n < number; n++)
    {
      pnl_vect_complex_set(ms, n, RCmul(-1, Conj(pnl_vect_complex_get(mj, n))));
      pnl_vect_complex_set(us, n, pnl_vect_complex_get(beta, n));
      pnl_vect_complex_set(mc, n, pnl_vect_complex_get(mj, 2 * number - 1 - n));
    }
```

```
  pnl_vect_complex_set(us, 0, RCmul(0.5, pnl_vect_complex_get(beta, 0)));
  for (n = number; n < 2 * number; n++)
    {
      pnl_vect_complex_set(ms, n, pnl_vect_complex_get(mj, 2 * number - n));
      pnl_vect_complex_set(us, n, CZERO);
      pnl_vect_complex_set(mc, n, pnl_vect_complex_get(mj, 2 * number - 1 - n));
    }
  pnl_vect_complex_set(ms, number, CZERO);

  pnl_fft_inplace(us);
  pnl_fft_inplace(ms);
  pnl_fft_inplace(mc);

  for (n = 0; n < 2 * number; n++)
    {
      pnl_vect_complex_set(ivector1, n, Cmul(pnl_vect_complex_get(ms, n), pnl_ve
      pnl_vect_complex_set(ivector2, n, Cmul(pnl_vect_complex_get(mc, n), pnl_ve
    }
  for (n = 0; n < number; n++)
    {
      pnl_vect_complex_set(ivector2, 2 * n + 1, RCmul(-1., pnl_vect_complex_get(
    }
  pnl_ifft_inplace(ivector1);
  pnl_ifft_inplace(ivector2);
  for (n = 0; n < number; n++)
    {
      pnl_mat_set(chat, domain, n, exp(-r * T / M) / M_PI * (Cimag(pnl_vect_comp
    }

  pnl_vect_complex_free(&ms);
  pnl_vect_complex_free(&mc);
  pnl_vect_complex_free(&us);
  pnl_vect_complex_free(&ivector1);
  pnl_vect_complex_free(&ivector2);
}

void static fc(double deltat, double x, double sigma, double r, double divid, do
{
  int k;
  double sum;
  PnlVectComplex *phi;
```

```
  sum = 0.0;
  phi = pnl_vect_complex_create(N);

  fphi(N, deltat, r, divid, sigma, x, a, b, phi);
  sum = sum + 0.5 * Creal(pnl_vect_complex_get(phi, 0)) * pnl_vect_get(ve, 0);

  for (k = 1; k < N; k++)
    {
      sum = sum + Creal(pnl_vect_complex_get(phi, k)) * pnl_vect_get(ve, k);
    }
  *smallc = exp(-r * deltat) * sum;
  pnl_vect_complex_free(&phi);
}


void static fV(int m, double a, double b, double k1, double k2, int N, int matur
{
  double deltat, xi1, xi2, xi12, psi1, psi2, psi12, sum, result;
  int k, i;
  PnlVect *ve;
  PnlMat *mm, *chat;
  PnlVectComplex *mj, *phi_min, *phi_max, *beta_min, *beta_max;

  deltat = T / maturity;
  beta_min = pnl_vect_complex_create(N);
  beta_max = pnl_vect_complex_create(N);
  phi_min = pnl_vect_complex_create(N);
  phi_max = pnl_vect_complex_create(N);
  mj = pnl_vect_complex_create(2 * N + 1);
  mm = pnl_mat_create(maturity + 1, N);
  ve = pnl_vect_create(N);
  chat = pnl_mat_create(Npoint + 1, N);

  if (m == maturity)
    {
      for (k = 0; k < N; k++)
        {
          fxi(a, b, log(k1), b, k, &xi1);
          fxi(a, b, log((k1 + k2) / 2.), b, k, &xi12);
          fxi(a, b, log(k2), b, k, &xi2);
```

```
        fpsi(a, b, log(k1), b, k, &psi1);
        fpsi(a, b, log((k1 + k2) / 2.), b, k, &psi12);
        fpsi(a, b, log(k2), b, k, &psi2);
        result = 2. / (b - a) * (xi1 - 2.*xi12 + xi2 - k1 * psi1 + (k1 + k2) *
        pnl_mat_set(mv, maturity, k, result);
      }
  }
else
  {
    sum = 0.0;
    pnl_mat_get_row(ve, mv, m + 1);
    mbasic(a, b, a, pnl_vect_get(midpoint, 1), N, mj);
    fphi(N, deltat, r, divid, sigma_min, a, a, b, phi_min);
    fphi(N, deltat, r, divid, sigma_max, a, a, b, phi_max);
    fbeta(N, phi_min, ve, beta_min);
    fbeta(N, phi_max, ve, beta_max);
    if (flag == 1)
      {
        c_fft(0, N, T, maturity, a, b, r, mj, beta_min, chat);
      }
    else
      {
        c_fft(0, N, T, maturity, a, b, r, mj, beta_max, chat);
      }

    for (i = 1; i < Npoint; i++)
      {
        flag = flag * (-1);
        mbasic(a, b, pnl_vect_get(midpoint, i), pnl_vect_get(midpoint, i + 1),
        if (flag == 1)
          {
            c_fft(i, N, T, maturity, a, b, r, mj, beta_min, chat);
          }
        else
          {
            c_fft(i, N, T, maturity, a, b, r, mj, beta_max, chat);
          }
      }
    flag = flag * (-1);
    mbasic(a, b, pnl_vect_get(midpoint, Npoint), b, N, mj);
    if (flag == 1)
```

```
        {
          c_fft(Npoint, N, T, maturity, a, b, r, mj, beta_min, chat);
        }
      else
        {
          c_fft(Npoint, N, T, maturity, a, b, r, mj, beta_max, chat);
        }
      for (k = 0; k < N; k++)
        {
          sum = 0.0;
          for (i = 0; i < Npoint + 1; i++)
            {
              sum = sum + pnl_mat_get(chat, i, k);
            }
          pnl_mat_set(mv, m, k, sum);
        }
    }
  pnl_vect_complex_free(&beta_min);
  pnl_vect_complex_free(&beta_max);
  pnl_vect_complex_free(&phi_min);
  pnl_vect_complex_free(&phi_max);
  pnl_vect_complex_free(&mj);
  pnl_mat_free(&mm);
  pnl_vect_free(&ve);
  pnl_mat_free(&chat);
}


void static ChangePoint(int m, double r, double divid, double sigma_min, double
{
  int i, J, ind, counter;
  double x, deltat, cmin, cmax;
  PnlVect *ve;

  ve = pnl_vect_create(N);
  deltat = T / M;
  J = (b - a) / 0.01;
  counter = 0;

  for (i = 0; i < N; i++)
    {
```

```
      pnl_vect_set(ve, i, pnl_mat_get(mv, m + 1, i));
    }
  fc(deltat, a, sigma_min, r, divid, a, b, N, ve, &cmin);
  fc(deltat, a, sigma_max, r, divid, a, b, N, ve, &cmax);
  if ((cmin - cmax) <= 0)
    {
      ind = 1;
      *flag = 1;
    }
  else
    {
      ind = -1;
      *flag = -1;
    }
  for (i = 1; i < J; i++)
    {
      x = a + i * 0.01;
      fc(deltat, x, sigma_min, r, divid, a, b, N, ve, &cmin);
      fc(deltat, x, sigma_max, r, divid, a, b, N, ve, &cmax);
      if ((cmin - cmax)*ind > 0.00001)
        {
          counter = counter + 1;
          pnl_vect_resize(midpoint, counter + 1);
          pnl_vect_set(midpoint, counter, x);
          ind = -1 * ind;

        }
    }
  *Npoint = counter;
  pnl_vect_free(&ve);
}

static int ap_uvmfouriercosine(double T, double k1, double k2, double s0, double
{
  int m, Npoint, flag;
  double x, a, b, cmin, cmax;
  PnlVect *midpoint, *ve;
  PnlMat *mv;


  m = 0;
```

```
  Npoint = 0;
  flag = 1;
  x = log(s0);
  a = log(70);
  b = log(130);
  midpoint = pnl_vect_create_from_zero(2);
  ve = pnl_vect_create(N);
  mv = pnl_mat_create(M + 1, N);

  fV(M, a, b, k1, k2, N, M, T, sigma_min, sigma_max, r, divid, Npoint, flag, mid
  pnl_mat_get_row(ve, mv, M);

  for (m = M - 1; m > 0; m--)
    {
      ChangePoint(m, r, divid, sigma_min, sigma_max, T, a, b, k1, k2, N, M, mv,
      fV(m, a, b, k1, k2, N, M, T, sigma_min, sigma_max, r, divid, Npoint, flag,
    }
  pnl_mat_get_row(ve, mv, 1);

  fc(T / M, x, sigma_min, r, divid, a, b, N, ve, &cmin);
  fc(T / M, x, sigma_max, r, divid, a, b, N, ve, &cmax);
  if (cmin < cmax)
    {
      *ptprice = cmin;
    }
  else
    {
      *ptprice = cmax;
    }
  pnl_vect_free(&midpoint);
  pnl_vect_free(&ve);
  pnl_mat_free(&mv);

  return OK;
}

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

```
  double K1, K2;

  K1 = (ptOpt->PayOff.Val.V_NUMFUNC_1)->Par[0].Val.V_PDOUBLE;
  K2 = (ptOpt->PayOff.Val.V_NUMFUNC_1)->Par[1].Val.V_PDOUBLE;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

  return ap_uvmfouriercosine(ptOpt->Maturity.Val.V_DATE, K1, K2, ptMod->S0.Val.V
}

static int CHK_OPT(AP_UVMFOURIERCOSINE)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "ButterflyEuro") == 0))
    return OK;
  return WRONG;
}

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

      Met->Par[0].Val.V_INT2 = 240;
      Met->Par[1].Val.V_INT2 = 800;
    }

  return OK;
}

PricingMethod MET(AP_UVMFOURIERCOSINE) =
{
  "AP_UVMFOURIERCOSINE",
  {{"Time Discretization Steps", INT2, {100}, ALLOW}, {"Step of Fourier Cosine E
  CALC(AP_UVMFOURIERCOSINE),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(AP_UVMFOURIERCOSINE),
```

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