Help

```
/*COS method for American Put Option, BS model*/
/*American option price is approximated by Bermudan option prices with
 * different number of exercise dates, in combination with 4-point
 * Richarson extrapolation*/
/*Developed by F.Fang, C.W.Oosterlee (2008), implemented by B.Zhang*/

#include <pnl/pnl_mathtools.h>
#include <pnl/pnl_complex.h>
#include <pnl/pnl_vector.h>
#include <pnl/pnl_fft.h>
#include <pnl/pnl_complex.h>
#include "bs1d_std.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2011+2) //The "#els

static int CHK_OPT(AP_Cosine_Amer)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_Cosine_Amer)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static void Valomega(int N, double a, double b, PnlVect *omega)
{
  int j;

  for (j = 0; j < N; j++)
    {
      pnl_vect_set(omega, j, ((double)j)*M_PI / (b - a));
    }

}

static void Valcf(int N, PnlVect *omega,  double c1, double c2, PnlVectComplex *
{
  int j;
```

```
  for (j = 0; j < N; j++)
    {
      double omegaj = pnl_vect_get(omega, j);
      pnl_vect_complex_set(cf, j, Cexp(CRsub(Complex(0, omegaj * c1), 0.5 * c2 *
    }
}

static void cf0(PnlVectComplex *cf)
{
  pnl_vect_complex_set_real(cf, 0, 0.5 * pnl_vect_complex_get_real(cf, 0));
  pnl_vect_complex_set_imag(cf, 0, 0.5 * pnl_vect_complex_get_imag(cf, 0));
}

static void VjtM(int N, double a, double b, double K, PnlVect *omega, PnlVect *V
{
  int j;

  for (j = 0; j < N; j++)
    {
      double omegaj = pnl_vect_get(omega, j);
      pnl_vect_set(V, j, (-pow((1 + pow(omegaj, 2)), -1) * (cos((-a)*omegaj) - e
    }
}

static void VjtM0(double a, double b, double K, PnlVect *V)
{
  pnl_vect_set(V, 0, (exp(a) - 1.0 - a) * (2.0 / (b - a))*K);
}

static void VecRe(int N, double r, double dt,  PnlVect *V,  PnlVect *omega,  Pnl
{
  int j;
  for (j = 0; j < N; j++)
    {
      double Vj = pnl_vect_get(V, j);
      double omegaj = pnl_vect_get(omega, j);
      dcomplex cfj = Cmul(pnl_vect_complex_get(cf, j), Cexp(Complex(0, (x - a) *

      pnl_vect_set(Re, j, exp(-r * dt)*Vj * Creal(cfj));
    }
```

```
}

static void VecRe1(int N, double r, double dt,  PnlVect *V,  PnlVect *omega,
                   PnlVectComplex *cf,  double x, double a,  PnlVect *Re1)
{
  int j;
  for (j = 0; j < N; j++)
    {
      double Vj = pnl_vect_get(V, j);
      double omegaj = pnl_vect_get(omega, j);
      dcomplex cfj = Cmul(Cmul(pnl_vect_complex_get(cf, j), Cexp(Complex(0, (x -

      pnl_vect_set(Re1, j, exp(-r * dt)*Vj * Creal(cfj));
    }
}

static void updatex(double K, double *f, double *fdelta, double *x)
{
  double g = 0;
  double gdelta = 0;

  if (*x < 0 || *x == 0)
    {
      g = K * (1 - exp(*x));
      gdelta = -K * exp(*x);
      *f = *f - g;
      *fdelta = *fdelta - gdelta;
    }
  *x = *x - *f / *fdelta;
}

static void updatexab(double *x, double a, double b)
{
  if (*x > b) *x = b;
  if (*x < a) *x = a;
}

static void Payoff(int N, PnlVect *omega, double x, double a, double b, double K
{
  int j;
```

```
  for (j = 0; j < N; j++)
    {
      double omegaj = pnl_vect_get(omega, j);
      pnl_vect_set(G, j, (-pow((1 + pow(omegaj, 2)), -1) * (cos((x - a)*omegaj)*
    }
}

static void Payoff0(double x, double a, double b, double K, double *G)
{
  *G = (exp(a) - exp(x) + x - a) * (2.0 / (b - a)) * K;
}
static void VecBasic(int N,  double x, double a, double b, PnlVectComplex *mj)
{
  int j;
  for (j = 0; j < N; j++)
    {
      pnl_vect_complex_set(mj, j, CRdiv(Csub(Cexp(Complex(0, j * M_PI)), Cexp(Co
    }
}

static void VecBasic0(double x, double a, double b, PnlVectComplex *mj)
{
  pnl_vect_complex_set(mj, 0, Complex(0, M_PI * (b - x) / (b - a)));
}

static void VecMs1(int N, PnlVectComplex *mj, PnlVectComplex *ms)
{
  int j;
  for (j = 0; j < N; j++)
    {
      dcomplex mjj = pnl_vect_complex_get(mj, j);

      pnl_vect_complex_set(ms, j, RCmul(-1, Conj(mjj)));
    }
}

static void VecMsN(int N, PnlVectComplex *ms)
{
  pnl_vect_complex_set(ms, N, Complex(0, 0));
}
```

```
static void VecMs2(int N, PnlVectComplex *mj, PnlVectComplex *ms)
{
  int j;

  for (j = N + 1; j < 2 * N; j++)
    {
      dcomplex mjj = pnl_vect_complex_get(mj, 2 * N - j);
      pnl_vect_complex_set(ms, j, mjj);
    }
}


static void VecPlus(int N, double x, double a, double b, PnlVectComplex *mjadd)
{
  int j;

  for (j = 0; j < N; j++)
    {
      pnl_vect_complex_set(mjadd, j, CRdiv(Csub(Cmul(Cexp(Complex(0, N * M_PI)),
    }
}

static void VecMc1(int N, PnlVectComplex *mjadd, PnlVectComplex *mc)
{
  int j;
  for (j = 0; j < N; j++)
    {
      dcomplex mja = pnl_vect_complex_get(mjadd, N - 1 - j);
      pnl_vect_complex_set(mc, j, mja);
    }
}

static void VecMc2(int N, PnlVectComplex *mj, PnlVectComplex *mc)
{
  int j;
  for (j = N; j < 2 * N; j++)
    {
      dcomplex mjj = pnl_vect_complex_get(mj, 2 * N - 1 - j);
      pnl_vect_complex_set(mc, j, mjj);
    }
}
```

```
static void VecUs1(int N, PnlVectComplex *cf, PnlVect *V, PnlVectComplex *us)
{
  int j;
  for (j = 0; j < N; j++)
    {
      double Vj = pnl_vect_get(V, j);
      dcomplex cfj = pnl_vect_complex_get(cf, j);
      pnl_vect_complex_set(us, j, RCmul(Vj, cfj));
    }
}

static void VecUs2(int N, PnlVectComplex *us)
{
  int j;
  for (j = N; j < 2 * N; j++)
    {
      pnl_vect_complex_set(us, j, Complex(0, 0));
    }
}

static void VecMul(int N, PnlVectComplex *vec1, PnlVectComplex *vec2, PnlVectCom
{
  pnl_vect_complex_clone(vec3, vec1);
  pnl_vect_complex_mult_vect_term(vec3, vec2);
}

static void VecSgn(int N, PnlVectComplex *vec)
{
  int j;
  for (j = 0; j < N; j++)
    {
      dcomplex vecj = RCmul(-1, pnl_vect_complex_get(vec, 2 * j + 1));
      pnl_vect_complex_set(vec, 2 * j + 1, vecj);
    }
}

static void MsuMcu(int N, PnlVectComplex *vec1, PnlVectComplex *vec2, PnlVectCom
{
  int j;
  for (j = 0; j < N; j++)
```

```
    {
      dcomplex vec1j = pnl_vect_complex_get(vec1, j);
      dcomplex vec2j = pnl_vect_complex_get(vec2, N - 1 - j);
      pnl_vect_complex_set(vec3, j, vec1j);
      pnl_vect_complex_set(vec4, j, vec2j);
    }
}

static void ConVal(int N,  PnlVectComplex *vec1,  PnlVectComplex *vec2,  PnlVect
{
  int j;
  for (j = 0; j < N; j++)
    {
      double vec1j = pnl_vect_complex_get_imag(vec1, j);
      double vec2j = pnl_vect_complex_get_imag(vec2, j);
      pnl_vect_set(C, j, exp(-r * dt) / M_PI * (vec1j + vec2j));
    }
}


static void VecAdd(int N, PnlVect *C, PnlVect *G, PnlVect *V)
{
  pnl_vect_clone(V, G);
  pnl_vect_plus_vect(V, C);
}

static void stepback(int N, int M, PnlVect *V, PnlVect *omega, PnlVectComplex *c
{

  PnlVect *Re, *Re1, *G, *C;
  PnlVectComplex *mj, *mjminus, *mjadd, *ms, *mc, *us, *ivector1, *ivector2, *Ms
  int m, j;
  double x, f, fdelta;

  Re = pnl_vect_create(N);
  Re1 = pnl_vect_create(N);
  G = pnl_vect_create(N);
  C = pnl_vect_create(N);

  mj = pnl_vect_complex_create(N);
  mjminus = pnl_vect_complex_create(N);
```

```
mjadd = pnl_vect_complex_create(N);
ms = pnl_vect_complex_create(2 * N);
mc = pnl_vect_complex_create(2 * N);
us = pnl_vect_complex_create(2 * N);
ivector1 = pnl_vect_complex_create(2 * N);
ivector2 = pnl_vect_complex_create(2 * N);
Msu = pnl_vect_complex_create(N);
Mcu = pnl_vect_complex_create(N);

x = 0.0;

//Backward Recursion

for (m = 1; m < M; m++)
  {
    // Locating the early-exercise point through Newton method
    // Error is of the order 1e-10 by five step. The default number of steps i

    for (j = 1; j < 6; j++)
      {
        VecRe(N, r, dt, V, omega, cf, x, a, Re);
        f = pnl_vect_sum(Re);
        VecRe1(N, r, dt, V, omega, cf, x, a, Re1);
        fdelta = pnl_vect_sum(Re1);
        updatex(K, &f, &fdelta, &x);
      }

    updatexab(&x, a, b);

    //compute V_k(t_m), m=M-1,...,1

    //compute G_k(t_m)

    Payoff(N, omega, x, a, b, K, G);
    Payoff0(x, a, b, K, pnl_vect_lget(G, 0));

    //compute C_k(t_m)

    VecBasic(N, x, a, b, mj);
    VecBasic0(x, a, b, mj);
```

```
    VecMs1(N, mj, ms);
    VecMsN(N, ms);
    VecMs2(N, mj, ms);

    VecPlus(N, x, a, b, mjadd);

    VecMc1(N, mjadd, mc);
    VecMc2(N, mj, mc);

    VecUs1(N, cf, V, us);
    VecUs2(N, us);

    // Three steps of forward FFT and two steps of backward FFT

    pnl_fft_inplace(us);

    pnl_fft_inplace(ms);

    pnl_fft_inplace(mc);

    VecMul(N, ms, us, ivector1);
    VecMul(N, mc, us, ivector2);
    VecSgn(N, ivector2);

    pnl_ifft_inplace(ivector1);
    pnl_ifft_inplace(ivector2);

    MsuMcu(N, ivector1, ivector2, Msu, Mcu);
    ConVal(N, Msu, Mcu, C, r, dt);

    // V_k(t_m)= G_k(t_m)+ C_k(t_m)

    VecAdd(N, C, G, V);
  }
// Option value, obtained from V_k(t_1)

x = log(S0 / K);
VecRe(N, r, dt, V, omega, cf, x, a, Re);
*vopt = pnl_vect_sum(Re);

pnl_vect_free(&Re);
```

```
  pnl_vect_free(&G);
  pnl_vect_free(&C);
  pnl_vect_complex_free(&mj);
  pnl_vect_complex_free(&mjminus);
  pnl_vect_complex_free(&mjadd);
  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);
  pnl_vect_complex_free(&Msu);
  pnl_vect_complex_free(&Mcu);
}

static int Cosine(double S0, double K, double T, double r, double q, double
                  sigma, int iscall, double *prix)
{
  /*  Values of N, M and L are chosen from the point of view of both speed
   *  and accuracy. Please do not change them.  */
  int N = 256;
  int L = 8;
  int M = 8; //Bermudan option values with 8, 16, 32 and 64 early exercise dates
  double  dt1, dt2, dt4, dt8, x, a, b, c11, c12, c14;
  double c18, c21, c22, c24, c28, Tmean, Tvar;
  PnlVect *omega, *V;
  double vopt1, vopt2, vopt4, vopt8;
  PnlVectComplex *cf1, *cf2, *cf4, *cf8;

  omega = pnl_vect_create(N);
  V = pnl_vect_create(N);
  cf1 = pnl_vect_complex_create(N);
  cf2 = pnl_vect_complex_create(N);
  cf4 = pnl_vect_complex_create(N);
  cf8 = pnl_vect_complex_create(N);

  /*Transform the stock price to log-asset domain: x=log(S/K)*/
  x = log(S0 / K);

  /*Distance between two consecutive exercise dates*/
  dt1 = T / ((double)M);
  dt2 = T / ((double)(2 * M));
```

```
dt4 = T / ((double)(4 * M));
dt8 = T / ((double)(8 * M));

/*Cumulants*/
c11 = (r - q - 0.5 * pow(sigma, 2)) * dt1;
c21 = pow(sigma, 2) * dt1;
c12 = (r - q - 0.5 * pow(sigma, 2)) * dt2;
c22 = pow(sigma, 2) * dt2;
c14 = (r - q - 0.5 * pow(sigma, 2)) * dt4;
c24 = pow(sigma, 2) * dt4;
c18 = (r - q - 0.5 * pow(sigma, 2)) * dt8;
c28 = pow(sigma, 2) * dt8;

/*Truncation range*/
Tmean = c11 * M;
Tvar = c21 * M;
a = Tmean - L * pow(Tvar, 0.5) + x;
b = Tmean + L * pow(Tvar, 0.5) + x;

Valomega(N, a, b, omega);

/*Characteristic function of Black-Scholes model*/
/*Characteristic function of Black-Scholes model*/

Valcf(N, omega, c11, c21, cf1);
cf0(cf1);
Valcf(N, omega, c12, c22, cf2);
cf0(cf2);
Valcf(N, omega, c14, c24, cf4);
cf0(cf4);
Valcf(N, omega, c18, c28, cf8);
cf0(cf8);

/* Fourier Cosine Coefficient of option price at expiry*/

VjtM(N, a, b, K, omega, V);

VjtM0(a, b, K, V);

/* Stepping back m = M-1,...1, to get V_j(t_m)
    The value of put option at t_0 is obtained from V_j(t_1)*/
```

```
  stepback(N, M, V, omega, cf1, r, dt1, a, b, S0, K, &vopt1);
  VjtM(N, a, b, K, omega, V);
  VjtM0(a, b, K, V);

  stepback(N, 2 * M, V, omega, cf2, r, dt2, a, b, S0, K, &vopt2);
  VjtM(N, a, b, K, omega, V);
  VjtM0(a, b, K, V);

  stepback(N, 4 * M, V, omega, cf4, r, dt4, a, b, S0, K, &vopt4);
  VjtM(N, a, b, K, omega, V);
  VjtM0(a, b, K, V);

  stepback(N, 8 * M, V, omega, cf8, r, dt8, a, b, S0, K, &vopt8);
  *prix = 1.0 / 21.0 * (vopt8 * 64.0 - vopt4 * 56.0 + vopt2 * 14.0 - vopt1);

  pnl_vect_free(&omega);
  pnl_vect_free(&V);
  pnl_vect_complex_free(&cf1);
  pnl_vect_complex_free(&cf2);
  pnl_vect_complex_free(&cf4);
  pnl_vect_complex_free(&cf8);


  return OK;
}

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

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

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
  Met->Res[1].Val.V_DOUBLE = 0.;
  return Cosine(ptMod->S0.Val.V_PDOUBLE,
```

```
                ptOpt->PayOff.Val.V_NUMFUNC_1->Par[0].Val.V_PDOUBLE,
                ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, r, divid,
                ptMod->Sigma.Val.V_PDOUBLE, iscall,
                &(Met->Res[0].Val.V_DOUBLE));
}


static int CHK_OPT(AP_Cosine_Amer)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "PutAmer") == 0))
    return OK;

  return  WRONG;
}

#endif

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->Par[0].Val.V_PDOUBLE = 0.1;
      Met->init = 1;
      Met->HelpFilenameHint = "ap_cosine_bs_amer";
    }
  return OK;
}

PricingMethod MET(AP_Cosine_Amer) =
{
  "AP_Cosine_Amer",
  {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(AP_Cosine_Amer),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(AP_Cosine_Amer),
  CHK_ok,
  MET(Init)
};
```