Help

```
extern "C" {
#include "
href../../../mod/kou1d/kou1d_std/kou1d_std_h_src.pdfkou1d_std.h"
}
#include<iostream>
#include<cmath>
#include"
href../../../common/math/ap_kou_model/functions_h_src.pdfmath/ap_kou_model/funct

extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
  static int CHK_OPT(AP_Kou_Am)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(AP_Kou_Am)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else
  static int Kou_Ap_Am(double S0, NumFunc_1  *P, double T, double r, double divi
  {
    double K = P->Par[0].Val.V_DOUBLE;
    long double x[12] = {sigma, lambda, p, lambdap, lambdam, S0, K, r, T, divid,

    if ((P->Compute) == &Call)
      {
        //On utilise la dualitÃ© (Farjado & Mordecki)
        temp = x[7];
        x[7] = x[9];
        x[9] = temp;
        temp = x[2];
        x[2] = (1 - x[2]) * x[4] / (1 + x[4]);
        q = temp * x[3] / (x[3] - 1);
        temp = x[3];
        x[3] = x[4] + 1;
        x[4] = temp - 1;
        temp = x[5];
        x[5] = x[6];
```

```
      x[6] = temp;
    }
  long double
  ksi = x[2] * x[3] / (x[3] - 1) + q * x[4] / (x[4] + 1) - 1,
  nu = (x[7] - x[9]) - sigma * sigma / 2 - lambda * ksi,
  eps = 1e-16;

  long double y[8] = {nu, sigma, lambda, x[2], x[3], x[4], x[7] / (1 - exp(-x[


  KCE G(y);

  dichotomie solG(G, -x[4] + eps, -eps, eps1);
  beta3 = solG.racine()[0];

  newton solG1(G, beta3, eps);
  beta3 = -solG1.racine()[0];
  long double x0 = -x[4] - 10;
  while (G.f(x0) < 0)
    x0 = x0 - 10;
  dichotomie solG2(G, x0, -x[4] - eps, eps1);
  beta4 = solG2.racine()[0];
  newton solG3(G, beta4, eps);
  beta4 = -solG3.racine()[0];

  long double C = beta3 * beta4 * (1 + x[4]);
  long double D = x[4] * (1 + beta3) * (1 + beta4);
  x[10] = C;
  x[11] = D;
//for(i=0;i<12;i++)
//cout <<  x[i] <<endl;
  amer_eq f(x);
  long double x1, x2;
  x1 = x[6] / 2;
  x2 = x[6] / (2 + eps);


  if (f.f(x1)*f.f(x2) > 0)
    {
      int i = 1;
      long double temp;
```

```
    if (f.f(x1) > 0)
      {
        x2 = (1 + un * i / 100.) * x[6] / 2.;
        while (f.f(x2) > 0)
          {
            temp = x2;
            x2 = (1 + un * i / 100.) * x[6];
            x1 = temp;
            i++;
          }
      }
    else
      {
        x1 = (1 + eps - un * i / 100.) * x[6] / (2. + eps);
        while (f.f(x1) < 0)
          {
            temp = x1;
            x1 = (1 + eps - un * i / 100.) * x[6] / (2. + eps);
            x2 = temp;
            i++;
          }

      }
  }

dichotomie solf(f, x1, x2, 1e-1);

long double v0 = solf.racine()[0];

long double EuP, dEuP, EuP0, dEuP0, cst1, cst2, dcst1, dcst2, cst11, cst21,

long double z[8];

z[0] = nu;
z[1] = sigma;
z[2] = lambda;
z[3] = x[2];
z[4] = x[3];
z[5] = x[4];
z[6] = log(x[6] / v0);
z[7] = T;
```

```
cst1 = psiVN(z);
dcst1 = dpsiVN(z) / x[6];
z[6] = log(x[6] / x[5]);
cst11 = psiVN(z);
if ((P->Compute) == &Put)
  dcst11 = -dpsiVN(z) / x[5];
else
  dcst11 = dpsiVN(z) / x[6];
z[0] = (x[7] - x[9]) + sigma * sigma / 2 - lambda * ksi;
z[2] = lambda * (ksi + 1);
z[3] = x[2] * x[3] / ((1 + ksi) * (x[3] - 1));
z[4] = x[3] - 1;
z[5] = x[4] + 1;
cst21 = psiVN(z);
if ((P->Compute) == &Put)
  dcst21 = -dpsiVN(z) / x[5];
else
  dcst21 = dpsiVN(z) / x[6];
z[6] = log(x[6] / v0);
cst2 = psiVN(z);
dcst2 = dpsiVN(z) / x[6];

EuP0 = x[6] * exp(-x[7] * T) * (1 - cst1) - v0 * exp(-x[9] * T) * (1 - cst2)
dEuP0 = exp(-x[7] * T) * (1 - cst1) - x[6] * exp(-x[7] * T) * dcst1 + v0 * e
EuP = x[6] * exp(-x[7] * T) * (1 - cst11) - x[5] * exp(-x[9] * T) * (1 - cst
if ((P->Compute) == &Put)
  dEuP = -x[6] * exp(-x[7] * T) * dcst11 - exp(-x[9] * T) * (1 - cst21) + x[
else
  dEuP = -x[6] * exp(-x[7] * T) * dcst11 + exp(-x[7] * T) * (1 - cst11) + x[
long double proba = 1.0L - cst1, A, B, dA, dB;
A = powl(v0, beta3) * (beta4 * x[6] - (1 + beta4) * (v0 * expl(-x[9] * T) +
B = powl(v0, beta4) * (beta3 * x[6] - (1 + beta3) * (v0 * expl(-x[9] * T) +
dA = powl(v0, beta3) * (beta4 - (1 + beta4) * dEuP0 + exp(-x[7] * T) * (1 -
dB = powl(v0, beta4) * (beta3 - (1 + beta3) * dEuP0 + exp(-x[7] * T) * (1 -


if (x[7] != 0)
  {
    if (x[5] >= v0)
      *ptPrice = EuP + A * powl(x[5], -beta3) + B * powl(x[5], -beta4);
    else
```

```
        *ptPrice = x[6] - x[5] * exp(-x[9] * T);
    }
  else
    {
      *ptPrice = EuP;
    }
  if (x[7] != 0)
    {
      if (x[5] >= v0)
        {
          if ((P->Compute) == &Put)
            *ptDelta = dEuP - beta3 * A * powl(x[5], -beta3 - 1) - beta4 * B *
          else
            *ptDelta = dEuP + dA * powl(x[5], -beta3) + dB * powl(x[5], -beta4
        }
      else
        {
          if ((P->Compute) == &Put)
            *ptDelta = -exp(-x[9] * T) + exp(-x[7] * T) * (1 - cst11);
          else
            *ptDelta = 1.0;
        }
    }
  else
    {
      *ptDelta = dEuP;
    }
  return OK;
}

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

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

  return  Kou_Ap_Am(ptMod->S0.Val.V_PDOUBLE, ptOpt->PayOff.Val.V_NUMFUNC_1, pt
}
```

```
  static int CHK_OPT(AP_Kou_Am)(void *Opt, void *Mod)
  {
    if ((strcmp(((Option *)Opt)->Name, "CallAmer") == 0) || (strcmp(((Option *)O
      return OK;

    return WRONG;
  }


#endif //PremiaCurrentVersion
  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {
    return OK;
  }

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