Help

```
#include  "
href../../../mod/bs1d/bs1d_pad/bs1d_pad_h_src.pdfbs1d_pad.h"
#include "
href../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2009+2) //The "#els
static int CHK_OPT(AP_FixedAsian_LordLow)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_FixedAsian_LordLow)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else
static void GaussLegendre_lord(double x1, double x2, double *x, double *w, int n
{

  int m;
  int j;
  int i;
  double z1, z, xm, xl, pp, p3, p2, p1;

  m  = (np + 1) / 2;
  xm = 0.5 * (x2 + x1);
  xl = 0.5 * (x2 - x1);

  for (i = 1; i <= m; i++)
    {
      z = cos(M_PI * (i - 0.25) / (np + 0.5));

      do
        {
          p1 = 1.0;
          p2 = 0.0;
          for (j = 1; j <= np; j++)
            {
              p3 = p2;
              p2 = p1;
```

```
            p1 = ((2.0 * j - 1.0) * z * p2 - (j - 1.0) * p3) / j;
          }
        pp = np * (z * p1 - p2) / (z * z - 1.0);
        z1 = z;
        z  = z1 - p1 / pp;
      }
    while (fabs(z - z1) > 0.00000001);

    x[i]     = xm - xl * z;
    x[np + 1 - i] = xm + xl * z;

    w[i]     = 2.0 * xl / ((1.0 - z * z) * pp * pp);
    w[np + 1 - i] = w[i];
  }

}

//calcule l'integrale d'une fonction à deux variables
static  double integrale2_lord(double a, double b, int n1, double y, double S0,
{
  double s = 0.;

  int i;

  double *x, *w;

  x = malloc((n1 + 1) * sizeof(double));
  w = malloc((n1 + 1) * sizeof(double));

  GaussLegendre_lord(a, b, x, w, n1);

  for (i = 1; i < (n1) + 1; i++)
    {
      s = s + w[i] * fct(x[i], y, S0, K, T, R, DIVID, SIGMA);
    }

  free(x);
  free(w);

  return s;
}
```

```
//trouve deux points dont l'image par fct est de signe différent pour pouvoir ap

static double  bornage_lord(double S0, double K, double T, double R, double DIVI

{
  double gauche = -1000;
  double droite = -1000;

  while (fct(gauche, S0, K, T, R, DIVID, SIGMA)*fct(droite, S0, K, T, R, DIVID,
    {
      gauche = gauche + 1;
    }

  return gauche;
}

/*dichotomie, recherche du zero d'une fonction*/
static double dichotomie_lord(double a, double b, double S0, double K, double T,
{
  double  gauche, droite, fg, fc, c;
  double precision = 0.00000001;


  /* Initialisations */
  gauche = a;
  droite = b;
  fg = fct(gauche, S0, K, T, R, DIVID, SIGMA) ;

  /* Boucle d'iteration */
  while ((droite - gauche) > precision)
    {
      c = (gauche + droite) / 2;

      fc = fct(c, S0, K, T, R, DIVID, SIGMA);
      if (fg * fc < 0)
        droite = c;
      else
        {
          gauche = c;
          fg = fc;
```

```
        }
    }

  return (gauche + droite) / 2.;

}

//fonction obtenu par les calculs sur l'article
static double g_lord(double t, double z, double S0, double K, double T, double R
{

  double  MU1 = log(S0) + ((R - DIVID - (pow(SIGMA, 2) / 2)) * T / 2);
  double A = (R - DIVID - pow(SIGMA, 2) / 2);

  return
    (S0 / T) * exp(A * t + ((3 * t / T) - 1.5 * pow(t / T, 2)) * (z - MU1) + 0.5
}



//integrale de g dont on retranche K
static double f_lord(double z, double S0, double K, double T, double R, double D
{
  return
    (integrale2_lord(0, T, 5 * T, z, S0, K, T, R, DIVID, SIGMA, g_lord) - K);
}

static double c2_lord(double t, double Lambda, double S0, double K, double T, do
{

  double  MU1 = log(S0) + ((R - DIVID - (pow(SIGMA, 2) / 2)) * T / 2);

  return (S0 / T) * exp((R - DIVID) * t) * cdf_nor(((SIGMA * t - 0.5 * SIGMA * p
}

//calcul le prix de l'option avec un conditionnement par la moyenne geometrique
static double prix_du_call1_lord(double Lambda, double S0, double K, double T, d
{
  double  MU1 = log(S0) + ((R - DIVID - (pow(SIGMA, 2) / 2)) * T / 2);

  return exp(-R * T) * (integrale2_lord(0, T, 5 * T, Lambda, S0, K, T, R, DIVID,
```

```
}


static double g1_lord(double t, double z, double S0, double K, double T, double

{
  double A = (R - DIVID - pow(SIGMA, 2) / 2);
  double  VAR2 = pow(S0 * SIGMA / T, 2) * pow(A, -3) * (-0.5 + 2 * exp(A * T) +
  double  MU2 = (S0 / (T * A)) * (exp(A * T) - 1);

  return (S0 / T) * exp(A * t + (pow(SIGMA, 2) * S0 / T) * ((1 - exp(A * t)) * p
}

//integrale de g1 dont on retranche K
static double f2_lord(double z, double S0, double K, double T, double R, double

{
  return
    (integrale2_lord(0, T, 4 * T, z, S0, K, T, R, DIVID, SIGMA, g1_lord)) - K;
}


static double c2_FA_lord(double t, double Lambda, double S0, double K, double T,
{
  double A = (R - DIVID - pow(SIGMA, 2) / 2);
  double  VAR2 = (pow(S0 * SIGMA / T, 2) * pow(A, -3) * (-0.5 + 2 * exp(A * T) +
  double  MU2 = (S0 / (T * A)) * (exp(A * T) - 1);



  return (S0 / T) * exp((R - DIVID) * t) * cdf_nor(((pow(SIGMA, 2) * S0 / T) * (
}

//  calcul le prix de l'option avec un conditionnement par l'approximation de ex
static  double prix_du_call2_lord(double Lambda, double S0, double K, double T,
{
  double A = (R - DIVID - pow(SIGMA, 2) / 2);
  double  VAR2 = (pow(S0 * SIGMA / T, 2) * pow(A, -3) * (-0.5 + 2 * exp(A * T) +
  double  MU2 = (S0 / (T * A)) * (exp(A * T) - 1);

  return exp(-R * T) * (integrale2_lord(0, T, 4 * T, Lambda, S0, K, T, R, DIVID,
```

```
}


static int LordLow_FixedAsian(double S0, double K, NumFunc_2  *po, double T, dou
{

  double inc;
  double CTtK, CTtK_inc, PTtK, Dlt, Plt;

  /*Increment for the Delta*/
  inc = 1.0e-3;

  if (flag == 1)
    {
      double b = bornage_lord(S0, K, T, R, DIVID, SIGMA, f_lord);
      double Lambda1 = dichotomie_lord(0, b, S0, K, T, R, DIVID, SIGMA, f_lord);

      /*Call Price */
      CTtK = prix_du_call1_lord(Lambda1, S0, K, T, R, DIVID, SIGMA);
      CTtK_inc = prix_du_call1_lord(Lambda1, S0 * (1 + inc), K, T, R, DIVID, SIG
    }
  else
    {
      double b2 = bornage_lord(S0, K, T, R, DIVID, SIGMA, f2_lord);
      double Lambda2 = dichotomie_lord(0, b2, S0, K, T, R, DIVID, SIGMA, f2_lord

      /*  Call Price */
      CTtK = prix_du_call2_lord(Lambda2, S0, K, T, R, DIVID, SIGMA);
      CTtK_inc = prix_du_call2_lord(Lambda2, S0 * (1 + inc), K, T, R, DIVID, SIG
    }

  /*  Put Price from Parity */
  if (R == DIVID)
    PTtK = CTtK + K * exp(-R * T) - S0 * exp(-R * T);
  else
    PTtK = CTtK + K * exp(-R * T) - S0 * exp(-R * T) * (exp((R - DIVID) * T) - 1

  /*Delta for call option*/
  Dlt = (CTtK_inc - CTtK) / (S0 * inc);;

  /*Delta for put option */
```

```
  if (R == DIVID)
    Plt = Dlt - exp(-R * T);
  else
    Plt = Dlt - exp(-R * T) * (exp((R - DIVID) * T) - 1.0) / (T * (R - DIVID));

  /*Price*/
  if ((po->Compute) == &Call_OverSpot2)
    *ptprice = CTtK;
  else
    *ptprice = PTtK;

  /*Delta */
  if ((po->Compute) == &Call_OverSpot2)
    *ptdelta = Dlt;
  else
    *ptdelta = Plt;

  return OK;
}

int CALC(AP_FixedAsian_LordLow)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  int return_value;
  double r, divid, time_spent, pseudo_spot, pseudo_strike;
  double t_0, T_0;

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

  T_0 = ptMod->T.Val.V_DATE;
  t_0 = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;


  if (T_0 < t_0)
    {
      Fprintf(TOSCREEN, "T_0 < t_0, untreated case\ n\ n\ n");
      return_value = WRONG;
    }
```

```
  /* Case t_0 <= T_0 */
  else
    {
      time_spent = (ptMod->T.Val.V_DATE - (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[
      pseudo_spot = (1. - time_spent) * ptMod->S0.Val.V_PDOUBLE;
      pseudo_strike = (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE - ti

      if (pseudo_strike <= 0.)
        {
          Fprintf(TOSCREEN, "ANALYTIC FORMULA\ n\ n\ n");
          return_value = Analytic_KemnaVorst(pseudo_spot, pseudo_strike, time_sp
        }
      else
        return_value = LordLow_FixedAsian(pseudo_spot, pseudo_strike, ptOpt->Pay
    }

  return return_value;
}
static int CHK_OPT(AP_FixedAsian_LordLow)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedEuro") == 0) || (strcmp(((Op
    return OK;
  return WRONG;
}
#endif //PremiaCurrentVersion

static PremiaEnumMember ComputationMethodLowMembers[] =
{
  { "Geometric mean", 1},
  { "Approximation", 2},
  { NULL, NULLINT }
};

static DEFINE_ENUM(ComputationMethodLow, ComputationMethodLowMembers);


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

```
      Met->Par[0].Val.V_ENUM.value = 1;
      Met->Par[0].Val.V_ENUM.members = &ComputationMethodLow;
    }

  return OK;
}

PricingMethod MET(AP_FixedAsian_LordLow) =
{
  "AP_FixedAsian_LordLow",
  { {"Conditioning Method", ENUM, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FOR
  CALC(AP_FixedAsian_LordLow),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(AP_FixedAsian_LordLow),
  CHK_ok,
  MET(Init)
};
```