Help

```
#include <stdlib.h>
#include  "
href../../../mod/bs1d/bs1d_pad/bs1d_pad_h_src.pdfbs1d_pad.h"

/* diffusion coefficient C(x,t) */
static double coef(double x, double r, double sig, double t, double del, double
{
  double z;
  double  drift = r - del;
  double  sigma2 = 0.5 * SQR(sig);

  if ((r - del) == 0.)
    {
      z = (x + t - T);
      z = sigma2 * SQR(z);
    }
  else
    {
      z = -T + x + (1. - exp(-drift * t)) / drift;
      z = sigma2 * SQR(z);
    }
  return z;
}

/*  right-hand-side R(x,t) of the PDE satisfied by the correction term */
static double fixe(double x, double r, double t, double sig, double del, double
{
  double eta, y;
  double  sigma2 = 0.5 * SQR(sig);
  double  drift = r - del;
  double  drift3 = pow(r - del, 3.);
  if ((r - del) == 0.)
    {
      eta = sig * sig * t * t * t / 6. + sigma2 * t * SQR(T) - sigma2 * T * SQR(
      y = (x - 2.*(T - t));
      y = y * sigma2 * (x);
      y = y * exp(-(x * x) / (4.*eta));
      y = y / (2.*sqrt(M_PI * eta));
    }
```

```
  else
    {
      eta = sigma2 * t * SQR(T) - 2 * T * sigma2 * (t / drift + (exp(-drift * t)
      y = x + 2.*(-T + (1 - exp(-(drift) * t)) / (drift));
      y = y * sigma2 * (x);
      y = y * exp(-((x) * (x)) / (4.*eta));

      y = y / (2.*sqrt(M_PI * eta));
    }
  return y;
}

/*  Computation of f0 */
static double f0(double x, double r, double t, double sig, double del, double T)
{
  double eta, y;
  double  sigma2;
  double  drift = r - del;
  double  drift3;


  sigma2 = 0.5 * SQR(sig);
  drift3 = pow(r - del, 3.0);

  if ((r - del) == 0.)
    {
      eta = sig * sig * t * t * t / 6. + sigma2 * t * SQR(T) - sigma2 * T * SQR(
      y = -x * (cdf_nor(-x / (sqrt(2.*eta)))) + sqrt(eta) / sqrt(M_PI) * exp(-x

    }
  else
    {
      eta = sigma2 * t * SQR(T) - 2 * T * sigma2 * (t / drift + (exp(-drift * t)

      y = -x * (cdf_nor(-x / (sqrt(2.*eta)))) + sqrt(eta) / sqrt(M_PI) * exp(-x
    }

  return y;
}

/*  derivative of f0 w.r.t. S */
```

```
static double Residu0(double x, double r, double t, double T, double sig, double
{
  double eta, z;
  double  sigma2 = 0.5 * SQR(sig);
  double  drift = r - del;
  double  drift3 = pow(r - del, 3.);

  if (r - del != 0.)
    {
      eta = sigma2 * t * SQR(T) - 2 * T * sigma2 * (t / drift + (exp(-drift * t)
      z = -x / T * (cdf_nor(-x / (sqrt(2.*eta)))) + sqrt(eta) / sqrt(M_PI) * exp

    }
  else
    {
      eta = sig * sig * t * t * t / 6. + sigma2 * t * SQR(T) - sigma2 * T * SQR(
      z = -x / T * (cdf_nor(-x / (sqrt(2.*eta)))) + sqrt(eta) / sqrt(M_PI) * exp
    }
  return z;
}

/*  tridiagonal matrix  */
static void Coef(double *U, double *D, double *L, double *G, double *Y, double *
{
  int i;
  double t2, coeff, coeff1;
  double *D1, *U1, *L1;
  double dt_dx = dt / (dx * dx);

  t2 = t + dt / 2;

  U1 = malloc((nb - 1) * sizeof(double));
  L1 = malloc((nb - 1) * sizeof(double));
  D1 = malloc(nb * sizeof(double));

  for (i = 0; i < nb - 1; i++)
    {
      coeff = coef(coorx[i], r, sig, t2, del, T);
      coeff1 = coef(coorx[i + 1], r, sig, t2, del, T);

      D[i] = 1 + dt_dx * coeff;
```

```
      L[i] = -0.5 * dt_dx * coeff1;
      U[i] = -0.5 * dt_dx * coeff;

      D1[i] = 1. - dt_dx * coeff;
      L1[i] = 0.5 * dt_dx * coeff1;
      U1[i] = 0.5 * dt_dx * coeff;
    }

  D[nb - 1] = 1. + dt_dx * coef(coorx[nb - 1], r, sig, t2, del, T);
  D1[nb - 1] = 1. - dt_dx * coef(coorx[nb - 1], r, sig, t2, del, T);

  G[0] = D1[0] * Y[0] + U1[0] * Y[1] + dt * fixe(coorx[0], r, t2, sig, del, T);
  for (i = 1; i < (nb - 1); i++)
    {
      G[i] = dt * fixe(coorx[i], r, t2, sig, del, T);
      G[i] = G[i] + L1[i - 1] * Y[i - 1] + D1[i] * Y[i] + U1[i] * Y[i + 1];
    }
  G[nb - 1] = dt * fixe(coorx[nb - 1], r, t2, sig, del, T) + L1[nb - 2] * Y[nb -

  free(D1);
  free(L1);
  free(U1);
}


/*   resolution of the system */
static void Gauss(double *X, double *L, double *U, double *D, double *G, int nb)
{
  int i;

  /*  BackWard Pass */
  for (i = nb - 2; i >= 0; i--)
    {
      D[i] = D[i] - U[i] * L[i] / D[i + 1];
      G[i] = G[i] - U[i] * G[i + 1] / D[i + 1];
    }

  /*  Forward Pass   */
  X[0] = G[0] / D[0];
  for (i = 1; i < nb; i++)
    {
```

```
      X[i] = (G[i] - L[i - 1] * X[i - 1]) / D[i];
    }

}


static void derivee(double *X, double *Y, int nbx, double dx)
{
  int i;
  for (i = 1; i < (nbx - 1); i++)
    {
      Y[i] = (X[i + 1] - X[i - 1]) / (2.*dx);
    }
}


/* correction  DELTA  */
#if 0
static double correction_DELTA(double f, double df, double x, double T, double t
{
  double cor;
  if (r - del == 0)
    cor = 1. / T * f - 1 / T * (x + t) * df;
  else
    cor = 1. / T * f - 1 / T * (x + 1 / (r - del) * (1 - exp(-(r - del) * t))) *

  return cor;
}
#endif

int Zhang_FloatingAsian(double pseudo_stock, double pseudo_strike, NumFunc_2  *p
{

  double CTtK, PTtK, Dlt, Plt;
  int k, i, p, nbx, nbt;
  double dx, dt, Xmin, prix, prix_exact, drift;
  double resi, resi_exact, correc_resi;
  double *Y, *Coorx, *Coort, *D, *L, *U, *G, *X, *DX;
  double xi, l, t;

  int pyr;
```

```
/*Discretization Time and Space Step Number*/
nbt = 100;
nbx = 100;

/*Memory Allocation*/
Coorx = malloc((nbx) * sizeof(double));
Coort = malloc(nbt * sizeof(double));
D = malloc(nbx * sizeof(double));
L = malloc((nbx) * sizeof(double));
U = malloc((nbx) * sizeof(double));
X = malloc(nbx * sizeof(double));
G = malloc(nbx * sizeof(double));
Y = malloc(nbx * sizeof(double));
DX = malloc((nbx) * sizeof(double));

for (i = 0; i < nbx; i++)
  Y[i] = 0.;

drift = r - divid;

dt = T / nbt;

/*  New variables xi and T */
if ((r - divid) == 0.)
  xi = 0.;
else
  xi = T - (1 - exp(-(drift) * T)) / (drift);

/*Localization */
l = 5.*sigma * pow(T, 1.5);
Xmin = -l;

/*****************************/
/* Discretization            */
/*****************************/
dx = 2.*l / (nbx + 1.);

Coorx[0] = Xmin + dx;
for (i = 1; i < nbx; i++)
  Coorx[i] = Coorx[i - 1] + dx;
```

```
Coort[0] = 0.;
for (k = 1; k < nbt; k++)  Coort[k] = Coort[k - 1] + dt;

/********************** Computation of the price  ***********************/

/* Approximate price   */
prix = exp(-divid * T) * pseudo_stock / T * f0(xi, r, T, sigma, divid, T);
/*  Computation of the correction   */

Coef(U, D, L, G, Y, Coorx, nbx, Coort[0], r, sigma, divid, dt, dx, T);
Gauss(X, L, U, D, G, nbx);

for (p = 1; p < nbt; p++)
  {
    t = Coort[p];
    Coef(U, D, L, G, X, Coorx, nbx, t, r, sigma, divid, dt, dx, T);
    Gauss(X, L, U, D, G, nbx);
  }

pyr = (int)floor((xi - Xmin) / dx);
prix_exact = prix + exp(-divid * T) * pseudo_stock / T * (X[pyr] + (X[pyr - 1]

/*  Put Price */
PTtK = prix_exact;

/*  Call Price from Parity*/
if (r == divid)
  CTtK = PTtK - pseudo_stock * exp(-r * t) + pseudo_stock * exp(-divid * t);
else
  CTtK = PTtK - pseudo_stock * exp(-r * t) * (exp((r - divid) * t) - 1.) / (t

/*Delta */

/* Computation of delta  */
resi = exp(-divid * T) * Residu0(xi, r, T, T, sigma, divid);

/*  correction of delta */
derivee(X, DX, nbx, dx);
correc_resi = 1 / T * (X[pyr] + (X[pyr - 1] - X[pyr]) * (xi - Coorx[pyr]) / (C
```

```
  correc_resi = exp(-divid * T) * correc_resi;

  resi_exact = resi + correc_resi;

  /*Delta for put option*/
  Plt = resi_exact;

  /*Delta for call option*/
  if (r == divid)
    Dlt = Plt - exp(-r * t) + exp(-divid * t);
  else
    Dlt = Plt - exp(-r * t) * (exp((r - divid) * t) - 1.) / (t * (r - divid)) +

  /*Price*/
  if ((po->Compute) == &Put_StrikeSpot2)
    *ptprice = PTtK;
  else
    *ptprice = CTtK;

  /*Delta */
  if ((po->Compute) == &Put_StrikeSpot2)
    *ptdelta = Plt;
  else
    *ptdelta = Dlt;

  /*Memory Desallocation*/
  free(Coorx);
  free(Coort);
  free(D);
  free(L);
  free(U);
  free(X);
  free(G);
  free(Y);
  free(DX);

  return OK;
}

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

```
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

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

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

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

  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
    {
      pseudo_strike = time_spent * (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[4].Val.
      return_value = Zhang_FloatingAsian(ptMod->S0.Val.V_PDOUBLE, pseudo_strike,
    }
  return return_value;

}

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

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

```
    {
      Met->init = 1;
    }

  return OK;
}

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