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 z;
  double  sigma2 = 0.5 * SQR(sig);

  if ((r - del) == 0.)
    {
      z = (x + t);
      z = sigma2 * SQR(z);
    }
  else
    {
      z = (x + (1 - exp((-r + del) * t)) / (r - del));
      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 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.;
      y = (x + 2.*t);
      y = y * sigma2 * x;
      y = y * exp(-(x * x) / (4.*eta));
      y = y / (2.*sqrt(M_PI * eta));
```

```
    }
  else
    {
      eta = (-3. + 2.*(r - del) * t + 4.*exp((-r + del) * t) - exp(-2 * (r - del
      eta = eta * sigma2 / (2 * drift3);
      y = (x + 2.*(1 - exp(-(r - del) * t)) / (r - del));
      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 eta, y;
  double  sigma2;
  double  drift3;


  sigma2 = SQR(sig);
  drift3 = pow(r - del, 3.0);
  if ((r - del) == 0.)
    {
      eta = sig * sig * t * t * t / 6.;
      y = -x / 2.*(1. + cdf_nor(-x / (2.*sqrt(eta)))) + sqrt(eta) / sqrt(M_PI) *
    }
  else
    {
      eta = (-3 + 2 * (r - del) * t + 4 * exp((-r + del) * t) - exp(-2.*(r - del
      eta = eta * sigma2 / (4.*drift3);
      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 to, double T, double sig, doubl
{
```

```
  double eta, z;
  double  sigma2 = SQR(sig);
  /* double  drift=r-del; */
  double  drift3 = pow(r - del, 3.);

  if ((r - del) == 0.)
    {
      eta = sig * sig * to * to * to / 6.;
      z = to / T * (cdf_nor(-x / (sqrt(2.*eta)))) + sqrt(eta / M_PI) / T * exp(-
    }
  else
    {
      eta = (-3 + 2 * (r - del) * to + 4 * exp((-r + del) * to) - exp(-2 * (r -
      eta = eta * sigma2 / (4.*drift3);
      z = (1 - exp(-(r - del) * to)) / ((r - del) * T) * (cdf_nor(-x / (sqrt(2.*

    }
  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);
      coeff1 = coef(coorx[i + 1], r, sig, t2, del);

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

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


  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  */
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;
}

int Zhang_FixedAsian(double pseudo_stock, double pseudo_strike, NumFunc_2  *po,
{

  double CTtK, PTtK, Dlt, Plt;
  int k, i, p, nbx, nbt;
  double dx, dt, Xmin, prix, prix_exact;
  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.;

dt = T / nbt;

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

/*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);

/*  Computation of the correction   */

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

/*  Call Price */
CTtK = prix_exact;

/*  Put Price from Parity*/
if (r == divid)
  PTtK = CTtK + pseudo_strike * exp(-r * t) - pseudo_stock * exp(-r * t);
else
  PTtK = CTtK + pseudo_strike * exp(-r * t) - pseudo_stock * exp(-r * t) * (ex

/*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 = correction_DELTA(X[pyr], DX[pyr], xi, T, T, r, divid);
  correc_resi = exp(-divid * T) * correc_resi;

  resi_exact = resi + correc_resi;

  /*Delta for call option*/
  Dlt = resi_exact;

  /*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;

  /*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_FixedAsian_Zhang)(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 = Zhang_FixedAsian(pseudo_spot, pseudo_strike, ptOpt->PayOf
    }
  return return_value;

}

static int CHK_OPT(AP_FixedAsian_Zhang)(void *Opt, void *Mod)
```

```
{
  if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedEuro") == 0) || (strcmp(((Op
    return OK;
  return WRONG;
}

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

  return OK;
}

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