Help

```
#include <stdlib.h>
#include  "
href../../../mod/bs1d/bs1d_std/bs1d_std_h_src.pdfbs1d_std.h"

static int npoints, n_emb;
static double h, k, pu, pm, pd, qu, quu, qd, qdd, z;
static double *P, *P1, *P2, *P3, *Q, *iv, *log_stock;
static double x, x_min, x_max, ln_stock;
static double K3_plus, K3_minus, K2_plus, K2_minus, K1_plus, K1_minus;
static double x3_max, x3_min, x2_max, x2_min, x1_max, x1_min;
static double node3_max, node3_min, node2_max, node2_min, node1_max, node1_min;
static double h1, h2, h3, k1, k2, k3;
static int npoints3, npoints2, npoints1;
static double epsilon = 1.0e-10;

/* AMM Algorith*/
/* Refinement near Strike Price at Maturity*/
void last_step(int am, double r, double K, double t, double z, NumFunc_1 *p)
{

  int i, j;
  /*Level 3 */
  /*Compute Domaine of Level 3*/
  j = 0;
  while (x_min + (double)j * h2 < log(K))
    j++;
  K3_minus = x_min + (j - 1) * h2;
  K3_plus = K3_minus + h2;
  x3_min = K3_plus - 4.*h2;
  x3_max = K3_minus + 4.*h2;
  node3_min = K3_plus - 2.*h2;
  node3_max = K3_minus + 2.*h2;

  /*Maturity Conditions Level 3*/
  ln_stock = x3_min;
  npoints3 = 15;
  for (i = 0; i < npoints3; i++)
    {
      iv[i] = (p->Compute)(p->Par, exp(ln_stock));
```

```
    P3[i] = iv[i];
    log_stock[i] = ln_stock;
    ln_stock += h3;
  }

/*Backward Resolution Level 3*/
for (i = 1; i <= 3; i++)
  {
    npoints3 -= 2;
    for (j = 0; j < npoints3; j++)
      {
        P3[j] = exp(-r * k3) * (pd * P3[j] + pm * P3[j + 1] + pu * P3[j + 2]);
        if (am)
          P3[j] = MAX((p->Compute)(p->Par, exp(log_stock[j + i] - z * k3 * (do
      }
  }

i = 4;
for (j = 0; j < 4; j++)
  {
    P3[j] = exp(-r * k3) * (pd * P3[2 * j] + pm * P3[2 * j + 1] + pu * P3[2 *
    if (am)
      P3[j] = MAX((p->Compute)(p->Par, exp(log_stock[i + 2 * j] - z * k3 * (do

  }

/*Level 2 */
/*Compute Domaine of Level 2*/
j = 0;
while (x_min + j * h1 < log(K))
  j++;
K2_minus = x_min + (j - 1) * h1;
K2_plus = K2_minus + h1;

x2_min = K2_plus - 4.*h1;
x2_max = K2_minus + 4.*h1;
node2_min = K2_plus - 2.*h1;
node2_max = K2_minus + 2.*h1;

/*Maturity Conditions Level 2*/
ln_stock = x2_min;
```

```
npoints2 = 15;
for (i = 0; i < npoints2; i++)
  {
    iv[i] = (p->Compute)(p->Par, exp(ln_stock));
    P2[i] = iv[i];
    log_stock[i] = ln_stock;
    ln_stock += h2;
  }

/*Backward Resolution Level 2*/
npoints2 = 15;
npoints2 -= 2;
n_emb = 0;
i = 1;
for (j = 0; j < npoints2; j++)
  {
    if ((x2_min + h2 + (double)j * h2) >= (node3_min - epsilon) && ((x2_min +
      {
        P2[j] = P3[n_emb];
        n_emb++;
      }
    else
      {
        P2[j] = exp(-r * k2) * (pd * P2[j] + pm * P2[j + 1] + pu * P2[j + 2]);
        if (am)
          P2[j] = MAX((p->Compute)(p->Par, exp(log_stock[j + i] - z * k2 * (do

      }
  }

for (i = 2; i <= 3; i++)
  {
    npoints2 -= 2;
    for (j = 0; j < npoints2; j++)
      {
        P2[j] = exp(-r * k2) * (pd * P2[j] + pm * P2[j + 1] + pu * P2[j + 2]);
        if (am)
          P2[j] = MAX((p->Compute)(p->Par, exp(log_stock[j + i] - z * k2 * (do

      }
  }
```

```
i = 4;
for (j = 0; j < 4; j++)
  {
    P2[j] = exp(-r * k2) * (pd * P2[2 * j] + pm * P2[2 * j + 1] + pu * P2[2 *

    if (am)
      P2[j] = MAX((p->Compute)(p->Par, exp(log_stock[i + 2 * j] - z * k2 * (do
  }


/*Level 1 */
/*Compute Domaine of Level 1*/
j = 0;
while (x_min + (double)j * h < log(K))
  j++;

K1_minus = x_min + (j - 1) * h;
K1_plus = K1_minus + h;
x1_min = K1_plus - 4.*h;
x1_max = K1_minus + 4.*h;
node1_min = K1_plus - 2.*h;
node1_max = K1_minus + 2.*h;


/*Maturity Conditions Level 1*/
npoints1 = 15;
ln_stock = x1_min;
for (i = 0; i < npoints1; i++)
  {
    iv[i] = (p->Compute)(p->Par, exp(ln_stock));
    P1[i] = iv[i];
    log_stock[i] = ln_stock;
    ln_stock += h1;
  }

/*Backward Resolution Level 1*/
npoints1 -= 2;
n_emb = 0;
i = 1;
for (j = 0; j < npoints1; j++)
```

```
  {
    if ((x1_min + h1 + (double)j * h1) >= (node2_min - epsilon) && ((x1_min +
      {
        P1[j] = P2[n_emb];
        n_emb++;
      }
    else
      {
        P1[j] = exp(-r * k1) * (pd * P1[j] + pm * P1[j + 1] + pu * P1[j + 2]);
        if (am)
          P1[j] = MAX((p->Compute)(p->Par, exp(log_stock[j + i] - z * k1 * (do

      }
  }

for (i = 2; i <= 3; i++)
  {
    npoints1 -= 2;
    for (j = 0; j < npoints1; j++)
      {
        P1[j] = exp(-r * k1) * (pd * P1[j] + pm * P1[j + 1] + pu * P1[j + 2]);
        if (am)
          P1[j] = MAX((p->Compute)(p->Par, exp(log_stock[j + i] - z * k1 * (do

      }
  }


i = 4;
for (j = 0; j < 4; j++)
  {
    P1[j] = exp(-r * k1) * (pd * P1[2 * j] + pm * P1[2 * j + 1] + pu * P1[2 *
    if (am)
      P1[j] = MAX((p->Compute)(p->Par, exp(log_stock[i + 2 * j] - z * k1 * (do
  }


/*Level 0*/
/*Maturity Conditions Level 0*/
ln_stock = x_min;
for (i = 0; i < npoints; i++)
```

```
    {
      iv[i] = (p->Compute)(p->Par, exp(ln_stock));
      log_stock[i] = ln_stock;
      P[i] = iv[i];
      ln_stock += h;

    }

  /*Backward Resolution Level 0*/
  npoints -= 2;
  n_emb = 0;
  i = 1;
  for (j = 0; j < npoints; j++)
    {
      if ((x_min + h + (double)j * h) >= (node1_min - epsilon) && ((x_min + h +
        {
          P[j] = P1[n_emb];
          n_emb++;
        }
      else
        {
          P[j] = exp(-r * k) * (pd * P[j] + pm * P[j + 1] + pu * P[j + 2]);
          if (am)
            P[j] = MAX((p->Compute)(p->Par, exp(log_stock[j + i] - z * k * (doub
        }
    }
}

/* AMM Algorith*/
static int FiglewskiGao(int am, double s, NumFunc_1 *p, double t, double r, doub
{
  double  K;
  int     i, j;


  npoints = 2 * N + 3;

  /*Price, intrinsic value arrays*/
  P = malloc(npoints * sizeof(double));
  Q = malloc(npoints * sizeof(double));
  P1 = malloc(npoints * sizeof(double));
```

```
P2 = malloc(npoints * sizeof(double));
P3 = malloc(npoints * sizeof(double));
iv = malloc(npoints * sizeof(double));
log_stock = malloc(npoints * sizeof(double));

/*Time and Space step Level ,0,1,2,3*/
k = t / ((double)N + 1. / 4. + 1. / 16.);
h = sigma * sqrt(3.*k);
h1 = h / 2.;
h2 = h / 4;
h3 = h / 8.;
k1 = k / 4.;
k2 = k / 16.;
k3 = k / 64.;

/*Discounted Probability*/
K = p->Par[0].Val.V_DOUBLE;
z = (r - divid) - SQR(sigma) / 2.;
x = log(s);
pu = 1. / 6.;
pm = 2. / 3.;
pd = 1. / 6.;
quu = 1. / 48.;
qdd = 1. / 48.;
qu = 23. / 48.;
qd = 23. / 48.;

/*Intrinsic value initialisation and terminal values*/
x_min = x - (double)(N + 1) * h + z * t;
x_max = x + (double)(N + 1) * h + z * t;

/*Last Step*/
last_step(am, r, K, t, z, p);

/*Backward Resolution*/
for (i = 2; i <= N - 1; i++)
  {
    npoints -= 2;
    for (j = 0; j < npoints; j++)
      {
        P[j] = exp(-r * k) * (pd * P[j] + pm * P[j + 1] + pu * P[j + 2]);
```

```
        if (am)
          P[j] = MAX((p->Compute)(p->Par, exp(log_stock[j + i] - z * k * (doub
      }
  }

/*Refinement near Stock Price for Delta Computation*/
/*step N*/
Q[0] = exp(-r * k) * (pd * P[0] + pm * P[1] + pu * P[2]);
Q[1] = exp(-r * k) * (qdd * P[0] + qd * P[1] + qu * P[2] + quu * P[3]);
Q[2] = exp(-r * k) * (pd * P[1] + pm * P[2] + pu * P[3]);
Q[3] = exp(-r * k) * (qdd * P[1] + qd * P[2] + qu * P[3] + quu * P[4]);
Q[4] = exp(-r * k) * (pd * P[2] + pm * P[3] + pu * P[4]);
if (am)
  for (j = 0; j < npoints; j++)
    Q[j] = MAX((p->Compute)(p->Par, exp(x - h + (double)j * h / 2. + z * (k1 +

/*step N+1/4*/
P[0] = exp(-r * k1) * (pd * Q[0] + pm * Q[1] + pu * Q[2]);
P[1] = exp(-r * k1) * (qdd * Q[0] + qd * Q[1] + qu * Q[2] + quu * Q[3]);
P[2] = exp(-r * k1) * (pd * Q[1] + pm * Q[2] + pu * Q[3]);
P[3] = exp(-r * k1) * (qdd * Q[1] + qd * Q[2] + qu * Q[3] + quu * Q[4]);
P[4] = exp(-r * k1) * (pd * Q[2] + pm * Q[3] + pu * Q[4]);
if (am)
  {
    for (j = 0; j < npoints; j++)
      P[j] = MAX((p->Compute)(p->Par, exp(x - h / 2. + (double)j * h / 4. + z
  }
/*step N+1/4+1/16*/
Q[0] = exp(-r * k2) * (qdd * P[0] + qd * P[1] + qu * P[2] + quu * P[3]);
Q[1] = exp(-r * k2) * (pd * P[1] + pm * P[2] + pu * P[3]);
Q[2] = exp(-r * k2) * (qdd * P[1] + qd * P[2] + qu * P[3] + quu * P[4]);

/*Delta*/
*ptdelta = (Q[2] - Q[0]) / (s * h2);

/*Price*/
*ptprice = Q[1];

/*Memory desallocation*/
free(P);
free(Q);
```

```
  free(P1);
  free(P2);
  free(P3);
  free(iv);
  free(log_stock);

  return OK;
}

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

  return  OK;
}

int CALC(TR_FiglewskiGao)(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 FiglewskiGao(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE, ptOpt->
                      ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, r, divid
                      Met->Par[0].Val.V_INT, &(Met->Res[0].Val.V_DOUBLE), &(Met-
}

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

```
      Met->Par[0].Val.V_INT2 = 100;


    }

  return OK;
}

PricingMethod MET(TR_FiglewskiGao) =
{
  "TR_FiglewskiGao",
  {{"StepNumber", INT2, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(TR_FiglewskiGao),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(TR_FiglewskiGao),
  CHK_tree,
  MET(Init)
};
```