Help

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

static int FSG_Asian(int type_asian, int am, double x, double y_1, double y_2, d
{
  double **C_n, **C_n_minus_one;
  int n, j, k;
  double h, u, d, pu, pd, a1;
  double price;
  double dY, dZ;
  int kfloor_p, kfloor_m;
  double epsilon_p, epsilon_m, psi_p, psi_m, expdy, asian_value = 0, spot_value,

  /*Parameter for the pathdep discretization*/
  oneoverrho *= (int)floor(sqrt(N));

  /*Memory Allocation*/
  C_n = malloc(sizeof(double *) * (2 * N + 1));
  if (C_n == NULL)
    return MEMORY_ALLOCATION_FAILURE;
  C_n_minus_one = malloc(sizeof(double *) * (2 * N + 1));
  if (C_n_minus_one == NULL)
    return MEMORY_ALLOCATION_FAILURE;

  for (j = -N; j <= N; j++)
    {
      C_n[N + j]  = malloc(sizeof(double) * (2 * N * oneoverrho + 1));
      if (C_n[N + j] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
      C_n_minus_one[N + j] = malloc(sizeof(double) * (2 * N * oneoverrho + 1));
      if (C_n_minus_one[N + j] == NULL)
        return MEMORY_ALLOCATION_FAILURE;
    }

  /* Up and Down factors */
  h  = T / (double)N;
```

```
dZ = sigma * sqrt(h);
dY = dZ / (double)oneoverrho;
expdy = exp(dY);
expdz = exp(dZ);

/* Discrete risk-neutral probability */
a1 = exp(h * (r - divid));
u  = exp(sigma * sqrt(h));
d  = 1. / u;

pu  = (a1 - d) / (u - d);
pd  = 1. - pu;
pu *= exp(-r * h) ;
pd *= exp(-r * h) ;

/*Ratio for the delta*/
if (type_asian == 1)
  delta_factor = x;
else
  delta_factor = y_1;

/*Intrisic value initialisation and terminal values*/
spot_value = x * exp(-(double)(N + 1) * dZ);

for (j = -N; j <= N; j++)
  {
    spot_value *= expdz;
    asian_value = y_1 * exp(-(double)(N * oneoverrho + 1) * dY);

    for (k = -N * oneoverrho; k <= N * oneoverrho; k++)
      {
        asian_value *= expdy;
        C_n[N + j][N * oneoverrho + k] = (p->Compute)(p->Par, spot_value, y_2
      }
  }

/*Backward resolution*/
for (n = N - 1; n > 0; n--)
  {
    spot_value = x * exp(-(double)(n + 1) * dZ);
```

```
  for (j = -n; j <= n; j++)
    {

      spot_value *= expdz;
      asian_value = y_1 * exp(-(double)(n * oneoverrho + 1) * dY);

      for (k = -n * oneoverrho; k <= n * oneoverrho; k++)
        {
          asian_value *= expdy;

          psi_p     = ((n + 1) * asian_value / y_1 + spot_value * expdz / x)
          psi_m     = ((n + 1) * asian_value / y_1 + spot_value / (x * expdz

          kfloor_p  = (int)floor(log(psi_p) / dY);
          kfloor_m  = (int)floor(log(psi_m) / dY);

          epsilon_p = (psi_p * exp(-kfloor_p * dY) - 1.0) / (expdy - 1.0);
          epsilon_m = (psi_m * exp(-kfloor_m * dY) - 1.0) / (expdy - 1.0);

          if ((N - n) % 2 == 1)
            {
              price = pu * ((1.  - epsilon_p) * C_n[N + j + 1][N * oneoverrh
            }
          else
            {

              price = pu * ((1.  - epsilon_p) * C_n_minus_one[N + j + 1][N *

            }
          if (am)
            price = MAX(price, (p->Compute)(p->Par, spot_value, y_2 + asian_
          if ((N - n) % 2 == 1)
            C_n_minus_one[j + N][k + N * oneoverrho] = price;
          else
            C_n[j + N][k + N * oneoverrho] = price;


        }
    }

}
```

```
  psi_p     = (1.0 + expdz) / 2.0;
  psi_m     = (1.0 + 1.0 / expdz) / 2.0;

  kfloor_p  = (int)floor(log(psi_p) / dY);
  kfloor_m  = (int)floor(log(psi_m) / dY);

  epsilon_p = (psi_p * exp(-kfloor_p * dY) - 1.0) / (expdy - 1.0);
  epsilon_m = (psi_m  * exp(-kfloor_m * dY) - 1.0) / (expdy - 1.0);


  /* First Step*/
  if (N  % 2 == 1)
    {
      *ptdelta = (((1.  - epsilon_p) * C_n[N + 1][N * oneoverrho + kfloor_p] + e
      *ptprice = pu * ((1.  - epsilon_p) * C_n[N + 1][N * oneoverrho + kfloor_p]
    }
  else
    {
      *ptdelta = (((1.  - epsilon_p) * C_n_minus_one[N + 1][N * oneoverrho + kfl
      *ptprice = pu * ((1.  - epsilon_p) * C_n_minus_one[N + 1][N * oneoverrho +
    }
  if (am)
    {
      *ptprice = MAX(*ptprice, (p->Compute)(p->Par, x, y_2 + asian_value));
    }

  /* Memory Desallocation */
  for (j = -N; j <= N; j++)
    {
      free(C_n[N + j]);
      free(C_n_minus_one[N + j]);
    }
  free(C_n);
  free(C_n_minus_one);

  return OK;
}
```

```
int CALC(TR_Asian_FSG)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid, time_spent, asian_spot, pseudo_spot, T_0, t_0, T;
  int return_value, type_asian;

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

  time_spent = (T_0 - t_0) / (T - t_0);
  asian_spot = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[4].Val.V_PDOUBLE * time_spe
  pseudo_spot = (1. - time_spent) * ptMod->S0.Val.V_PDOUBLE;

  if (T_0 < t_0)
    {
      return_value = 0;
    }
  else
    {
      if (((ptOpt->PayOff.Val.V_NUMFUNC_2)->Compute == Call_StrikeSpot2) ||
          ((ptOpt->PayOff.Val.V_NUMFUNC_2)->Compute == Put_StrikeSpot2))
        /*Floating Case*/
        type_asian = 1;
      else type_asian = 0;

      return_value = FSG_Asian(type_asian, ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.V
    }

  return return_value;

}

static int CHK_OPT(TR_Asian_FSG)(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;
      Met->Par[0].Val.V_INT2 = 50;
      Met->Par[1].Val.V_INT2 = 2;
    }

  return OK;
}


PricingMethod MET(TR_Asian_FSG) =
{
  "TR_Asian_FSG",
  {{"StepNumber", INT2, {100}, ALLOW}, {"Inverse of Rho", INT2, {100}, ALLOW}, {
  CALC(TR_Asian_FSG),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(TR_Asian_FSG),
  CHK_tree,
  MET(Init)
};
```