Help

```
#include <stdlib.h>
#include  "
href../../../mod/bs1d/bs1d_pad/bs1d_pad_h_src.pdfbs1d_pad.h"
#define PRECISION 1.0e-7 /*Precision for the localization of FD methods*/

static int ExplicitLookback(int min_or_max, int am, double s, double s_max_min,
{
  int M, Index, TimeIndex, i, j;
  double **P, **Obst, **G, *vect_s, *vect_z;
  double h, z, k, p1, p2, p3, l, x, vv, upwind_alphacoef;

  /*Peclet Condition-Coefficient of diffusion augmente*/
  vv = 0.5 * SQR(sigma);
  z = (r - divid) - vv;
  l = sigma * sqrt(t) * sqrt(log(1.0 / PRECISION)) + fabs(z * t);

  h = 2.*l / (double)N;
  if ((h * fabs(z)) <= vv)
    upwind_alphacoef = 0.5;
  else
    {
      if (z > 0.) upwind_alphacoef = 0.0;
      else  upwind_alphacoef = 1.0;
    }
  vv -= z * h * (upwind_alphacoef - 0.5);
  k = SQR(h) / (2.*vv + r * SQR(h));
  M = (int)(t / k);
  x = log(s);

  /*Memory Allocation*/
  P = (double **)calloc(N + 1, sizeof(double *));
  for (i = 0; i < N + 1; i++)
    {
      P[i] = (double *)calloc(N + 1, sizeof(double));
    }

  Obst = (double **)calloc(N + 1, sizeof(double *));
```

```
for (i = 0; i < N + 1; i++)
  {
    Obst[i] = (double *)calloc(N + 1, sizeof(double));
  }

G = (double **)calloc(N + 1, sizeof(double *));
for (i = 0; i < N + 1; i++)
  {
    G[i] = (double *)calloc(N + 1, sizeof(double));
  }

vect_s = (double *)calloc(N + 1, sizeof(double));
vect_z = (double *)calloc(N + 1, sizeof(double));

for (i = 0; i <= N; i++)
  {
    vect_s[i] = x - l + (double)i * h;
    vect_z[i] = vect_s[i];
  }

/*"Probabilities" associated to points*/
p1 = k * (vv / (SQR(h)) - z / (2.0 * h));
p2 = 1.0 - k * (2.*vv / SQR(h) + r);
p3 = k * (vv / (SQR(h)) + z / (2.0 * h));

/*Maturity Condition*/
for (j = N; j >= 1; j--)
  for (i = 1; i < N; i++)
    {
      if (min_or_max == 0)
        P[i][j] = (p->Compute)(p->Par, exp(vect_s[i]), MAX(s_max_min, exp(vect
      else if (min_or_max == 1)
        P[i][j] = (p->Compute)(p->Par, exp(vect_s[i]), MIN(s_max_min, exp(vect
      Obst[i][j] = P[i][j];
    }

/*Finite Difference Cycle */
for (TimeIndex = 1; TimeIndex <= M; TimeIndex++)
  {
    /*Store*/
    for (i = 1; i <= N; i++)
```

```
      for (j = 1; j <= N; j++)
        G[i][j] = P[i][j];

    /*MAX CASE*/
    if (min_or_max == 0)
      {
        /*Compute for each level Z-Sup*/
        for (j = N - 1; j > 1; j--)
          for (i = 1; i < j; i++)
            {
              P[i][j] = p1 * G[i - 1][j] + p2 * G[i][j] + p3 * G[i + 1][j];
              if (am)
                P[i][j] = MAX(Obst[i][j], P[i][j]);
            }

        /*Neumann Derivative Approximation*/
        P[N - 1][N - 1] = P[N - 1][N];
        for (j = N - 2; j >= 1; j--)
          P[j][j] = (4.*P[j][j + 1] - P[j][j + 2]) / 3.;
      }
    else
      /*MIN CASE*/
      {
        /*Compute for each level Z-Inf*/
        for (j = 1; j < N; j++)
          for (i = j + 1; i < N; i++)
            {
              P[i][j] = p1 * G[i - 1][j] + p2 * G[i][j] + p3 * G[i + 1][j];
              if (am)
                P[i][j] = MAX(Obst[i][j], P[i][j]);
            }
        /*Neumann Derivative Approximation*/
        P[1][1] = P[1][0];
        for (j = N - 1; j >= 2; j--)
          //P[j][j]=P[j][j-1];
          P[j][j] = (4.*P[j][j - 1] - P[j][j - 2]) / 3.;
      }

  }
Index = (int)((double)N / 2.0);
```

```
  /*Price*/
  *ptprice = P[Index][Index];

  /*Delta*/
  *ptdelta = (P[Index + 1][Index + 1] - P[Index - 1][Index - 1]) / (2.*s * h);

  /*Memory desallocation*/
  for (i = 0; i < N + 1; i++)
    free(P[i]);
  free(P);

  for (i = 0; i < N + 1; i++)
    free(Obst[i]);
  free(Obst);

  for (i = 0; i < N + 1; i++)
    free(G[i]);
  free(G);

  free(vect_s);
  free(vect_z);


  return OK;
}

int CALC(FD_ExplicitLookback)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid;
  int  minormax;

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

  if ((ptOpt->MinOrElse).Val.V_BOOL == MAXIMUM)
    minormax = 0;
  else minormax = 1;

  return  ExplicitLookback(minormax, ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_P
```

```
}

static int CHK_OPT(FD_ExplicitLookback)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "LookBackPutFloatingEuro") == 0) || (strcmp
      || (strcmp(((Option *)Opt)->Name, "LookBackCallFloatingAmer") == 0) || (st
    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 = 200;
    }

  return OK;
}

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