Help

```
#include <stdlib.h>
#include  "bs1d_pad.h"
#include "error_msg.h"

static int  Babbs_95_FloatingCall(int am, double s, double s_min, NumFunc_2 *p,
{
  int i, j, eta0, npoints;
  double u, d, h, pu, pd, a1, stock, eta, y_0;
  double *P, *iv, *Q, *Boundary;
  double upperstock;
  int odd;
  double flat_price = 0., up_price;

  /*Price, intrisic value arrays*/
  P = malloc((2 * N + 1) * sizeof(double));
  if (P == NULL)
    return MEMORY_ALLOCATION_FAILURE;
  Q = malloc((2 * N + 1) * sizeof(double));
  if (Q == NULL)
    return MEMORY_ALLOCATION_FAILURE;
  iv = malloc((2 * N + 1) * sizeof(double));
  if (iv == NULL)
    return MEMORY_ALLOCATION_FAILURE;
  Boundary = malloc((N + 1) * sizeof(double));
  if (Boundary == NULL)
    return MEMORY_ALLOCATION_FAILURE;


  /*Up and Down factors*/
  y_0 = s_min / s;
  h = t / (double)N;
  a1 = exp(h * (r - divid));
  u = exp(sigma * sqrt(h));
  d = 1. / u;

  /*Critical Index*/
  eta = -log(y_0) / (sigma * sqrt(h));
  eta0 = (int)floor(eta);
  if (eta0 > N)
```

```
  eta0 = N;

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

if (eta0 < N)
  {
    /*First Stage:Computation of price value along the line spot=maximum*/

    /*Intrisic value initialisation*/
    for (i = 0; i <= N; i++)
      Boundary[i] = 0.;

    stock = 1.;
    for (i = 0; i < N + eta0; i++)
      {
        iv[i] = (p->Compute)(p->Par, 1., stock);
        P[i] = iv[i];
        Q[i] = P[i];
        stock *= d;
      }

    Boundary[N] = P[0];

    /*Backward Resolution*/
    for (i = 1; i < N - eta0; i++)
      {
        P[0] = pd * Q[0] + pu * Q[1];
        if (am)
          P[0] = MAX(iv[0], P[0]);
        Boundary[N - i] = P[0];

        for (j = 1; j <= N - eta0 - i; j++)
          {
            P[j] = pd * Q[j - 1] + pu * Q[j + 1];
            if (am)
              P[j] = MAX(iv[j], P[j]);
          }
```

```
        for (j = 0; j <= N - eta0 - i; j++)
          Q[j] = P[j];
      }
  }

  if (s_min < s)
    {
      /*Second Stage:Computation of price value */
      /*Intrinsic value initialization*/
      upperstock = y_0;
      for (i = 0; i < N; i++)
        upperstock *= d;
      stock = upperstock;
      for (i = 0; i <= N + eta0; i++)
        {
          iv[i] = (p->Compute)(p->Par, 1., stock);
          stock *= u;
        }

      /*Terminal Values*/
      npoints = eta0 + (N - eta0) / 2;
      for (j = 0; j <= npoints; j++)
        P[j] = iv[2 * j];

      /*Backward Resolution*/
      if (eta0 > 0) /*The first mesh does not breach the barrier*/
        {

          /*First part-the barrier is active*/
          odd = 1;
          for (i = eta0; i < N - 1; i++)
            odd = !odd;

          if (!odd) npoints = npoints - 1;

          for (i = 1; i <= N - eta0; i++)
            {
              for (j = 0; j < npoints; j++)
                {
                  P[j] = pu * P[j] + pd * P[j + 1];
```

```
        if (am)
          P[j] = MAX(iv[i + 2 * j], P[j]);
      }
    /*Special handling of the critical node*/
    if (odd)
      {
        P[npoints] = pu * P[npoints] + pd * Boundary[N + 1 - i];
        if (am)
          P[npoints] = MAX(iv[i + 2 * npoints], P[npoints]);
      }

    /*For the critical node at the next iteration*/
    if (!odd)
      {
        npoints = npoints - 1;
      }
    odd = !odd;
  }

/*Second part-the barrier is strictly below the tree*/
npoints = eta0 - 1;
for (i = N - eta0 + 1; i < N; i++)
  {
    for (j = 0; j <= npoints; j++)
      {
        P[j] = pu * P[j] + pd * P[j + 1];
        if (am)
          P[j] = MAX(iv[i + 2 * j], P[j]);
      }

    npoints = npoints - 1;
  }
/*Delta*/
*ptdelta = (P[1] - P[0]) / (y_0 * (u - d));

/*First time step*/
P[0] = pu * P[0] + pd * P[1];
if (am)
  P[0] = MAX(iv[N], P[0]);
/*Price*/
*ptprice = P[0];
```

```
    }
  else /*eta0=0, the first mesh breaches the barrier*/
    {
      /*The barrier is always active*/
      odd = 1;
      for (i = eta0; i < N - 1; i++)
        odd = !odd;

      if (!odd) npoints = npoints - 1;

      for (i = 1; i <= N - eta0 - 1; i++) /*We go backward until the next da
        {
          flat_price = P[1]; /*Only for the delta*/

          for (j = 0; j < npoints; j++)
            {
              P[j] = pu * P[j] + pd * P[j + 1];
              if (am)
                P[j] = MAX(iv[i + 2 * j], P[j]);
            }
          /*Special handling of the critical node*/
          if (odd)
            {
              P[npoints] = pu * P[npoints] + pd * Boundary[N + 1 - i];
              if (am)
                P[npoints] = MAX(iv[i + 2 * npoints], P[npoints]);;
            }


          if (!odd)
            {
              npoints = npoints - 1;
            }
          odd = !odd;

        }
      up_price = P[0]; /*For the delta*/

      /*First time step*/
      P[0] = pu * P[0] + pd * P[1];
```

```
        /*Special handling of the critical node*/
        P[0] = pu * P[0] + pd * Boundary[1];
        if (am)
          P[0] = MAX(iv[N], P[0]);

        /*Delta*/
        /*Corresponds to setting a third point at level s between u*s and d*s*
        /*One computes the finite difference approximation between s and us*/
        *ptdelta = (up_price - (exp(-r * h) * flat_price + exp(r * h) * P[0])
      } /*eta0=0*/
    }

  else /*s=s_min*/
    *ptdelta = (P[1] - P[0]) / (d - 1.);

  /*Price*/
  *ptprice = s * P[0];

  /*Delta*/
  *ptdelta = (*ptdelta) * (-y_0) + P[0];

  /*Memory Desallocation*/
  free(P);
  free(Q);
  free(iv);
  free(Boundary);

  return OK;

}

int CALC(TR_Babbs_Call)(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  Babbs_95_FloatingCall(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBL
```

```
}

static int CHK_OPT(TR_Babbs_Call)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "LookBackCallFloatingEuro") == 0) || (strcm
    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 = 1000;

    }

  return OK;
}

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