Help

```
#include "
href../../../mod/bharchiarella1d/bharchiarella1d_stdi/bharchiarella1d_stdi_h_src
#include "
href../../../common/error_msg_h_src.pdferror_msg.h"

static double *Pn, *Pnn, *Pnnn;

/*************************************************************************/
static double f0(double t, double beta0, double beta1, double eta)

{

  return (beta0 + beta1 * (1 - exp(-eta * t)));

}
/*************************************************************************/
/*static double f0_cf(double t,double beta0,double beta1,double eta)
{
  return(beta0+beta1*(1-exp(-eta*t)));
}*/
/*************************************************************************/

static double f2(double t, double beta1, double eta)

{

  return (beta1 * eta * exp(-eta * t));

}

/*************************************************************************/

static double D(double t, double beta0, double beta1, double eta, double lambda)

{

  return (f2(t, beta1, eta) + lambda * f0(t, beta0, beta1, eta));

}
```

```
/*************************************************************************/

/*static double alpha(double t, double tau_alpha,double lambda)
{
  return (exp(-lambda*t)/(exp(-lambda*tau_alpha)-exp(-lambda*t)));
}
*/
/*************************************************************************/

static double psi(double t, double x, double y, double lambda, double tau, doubl

{
  if (t > 0)
    {
      if (t < tau)
        {
          return (

                   lambda * exp(-lambda * t) /
                   (exp(-lambda * tau) - exp(-lambda * t)) * (x - f0(t, beta0, b
                   lambda * exp(lambda * (tau - t)) *
                   exp(-lambda * t) /
                   (exp(-lambda * tau) - exp(-lambda * t)) * (y - (beta0 + beta1
          /*
          return(max(lambda*exp(-lambda*t)/(exp(-lambda*tau)-
          exp(-lambda*t))*(x-f0(t))-
          lambda*exp(lambda*(tau-t))*exp(-lambda*t)/(exp(-lambda*tau)-
          exp(-lambda*t))*(y-(beta0+beta1*(1-exp(-eta*tau))) ),0));
          */
        }
      else
        {
          return (0.0);
        }
    }
  else
    {
      return (0.0);
    }
```

```
}

/*************************************************************************/

static double mur(double t, double x, double y, double lambda, double beta0, dou

{

  return (D(t, beta0, beta1, eta, lambda) + psi(t, x, y, lambda, tau, beta0, bet


}

/*************************************************************************/

static double sigmar(double x, double y, double gamma0, double alpha0, double al

{

  return (exp(gamma0 * log(alpha0 + alphar * x + alphaf * y)));

}

/*************************************************************************/

static double sigma1(double t, double x, double y, double lambda, double tau, do

{

  return (exp(-lambda * tau + lambda * t) * sigmar(x, y, gamma0, alpha0, alphar,

}

/*************************************************************************/

static double mu1(double t, double x, double y, double tau, double lambda, doubl

{
  return (sigma1(t, x, y, lambda, tau, gamma0, alpha0, alphar, alphaf) * sigma1(
          (exp(lambda * (tau - t)) - 1) / lambda);
```

```
}

/*************************************************************************/
/*static double beta(double t,double T,double lambda)
{

  return(1/lambda*(1-exp(-lambda*(T-t))));
}*/

/*resolution of LUx=B*/
/*********************************************************/
static void soLU(int ndr, double **A, double *B)
{
  double *Y;
  int i, N;


  N = ndr * ndr;

  if ((Y = (double *)calloc(N, sizeof(double))) == NULL)
    {
      printf("Impossible d'allouer le tableau Y\ n");
      exit(1);
    }
  /* resolution de LUx=B ou L et U sont issues de la facto incomp ILU(0) de A */
  /* et stockées dans A sauf la diago de 1 de L */

  /* initialisation */

  for (i = 0; i < N; i++)
    {

      Y[i] = 0;
    }

  /* resoudre LUX=B */

  /* Resoudre LY=B par descente triangulaire */

  Y[0] = B[0];
```

```
for (i = 1; i < ndr - 1; i++)
  {
    Y[i] = B[i];


    Y[i] -= A[i][1] * Y[i - 1];

  }

i = ndr - 1;

Y[i] = B[i];

Y[i] -= A[i][1] * Y[i - 1];

i = ndr;

Y[i] = B[i];

Y[i] -= A[i][3] * Y[i - ndr] + A[i][1] * Y[i - 1];

for (i = ndr + 1; i < N; i++)
  {

    Y[i] = B[i];

    Y[i] -= A[i][5] * Y[i - ndr - 1] + A[i][3] * Y[i - ndr] + A[i][1] * Y[i -



  }

/* Resoudre UX=Y par remontée triangulaire */


/* X est remplacé par B */

B[N - 1] = Y[N - 1] / A[N - 1][0];

for (i = N - 2; i > N - ndr; i--)
  {
```

```
      B[i] = Y[i] - A[i][2] * B[i + 1];

      B[i] /= A[i][0];
    }

  i = N - ndr;

  B[i] = Y[i] - A[i][2] * B[i + 1];

  B[i] /= A[i][0];
  i = N - ndr - 1;

  B[i] = Y[i] - A[i][4] * B[i + ndr] - A[i][2] * B[i + 1];

  B[i] /= A[i][0];

  for (i = N - ndr - 2; i > -1; i--)
    {

      B[i] = Y[i] - A[i][6] * B[i + ndr + 1] - A[i][4] * B[i + ndr] - A[i][2] *

      B[i] /= A[i][0];


    }
  free(Y);

  return;
}

/*Preconditioner GMRES Solver*/
static int resolution(int ndr, double **m, double *Pn, double *Pnn, double *Pnnn
{
  int ip, I, J, I1, I2, I3, I4;
  double tem, dem, hii, hipi, gamm, coss, sinn, hipj, res, hij, raux, som;
  int it, i, i1, k, j, j11, nk, nit, nkrMax, nkr, N;
  double erreur;
  double **A, *aux, *aux1, **v, **hh, *rr, *vec, *x;
  int iterr ;

  N = ndr * ndr;
```

```
iterr = 0;

nk = 1;
nit = 3;
nkrMax = 1;
nkr = 1;

if ((aux = (double *)calloc(N + 1, sizeof(double))) == NULL)
  {
    printf("Impossible d'allouer le tableau aux\ n");
    exit(1);
  }


if ((aux1 = (double *)calloc(N + 1, sizeof(double))) == NULL)
  {
    printf("Impossible d'allouer le tableau aux1\ n");
    exit(1);
  }
v = (double **)calloc(N + 1, sizeof(double *));
for (i = 0; i < N + 1; i++)
  {
    if ((v[i] = (double *)calloc(nkrMax + 1, sizeof(double))) == NULL)
      {
        printf("Impossible d'allouer le tableau v\ n");
        exit(1);
      }
  }
hh = (double **)calloc(nkrMax + 1, sizeof(double *));
for (i = 0; i < nkrMax + 1; i++)
  {
    if ((hh[i] = (double *)calloc(nkrMax + 1, sizeof(double))) == NULL)
      {
        printf("Impossible d'allouer le tableau h\ n");
        exit(1);
      }
  }
if ((rr = (double *)calloc(nkrMax + 1, sizeof(double))) == NULL)
  {
    printf("Impossible d'allouer le tableau rr\ n");
    exit(1);
```

```
  }
if ((vec = (double *)calloc(N + 1, sizeof(double))) == NULL)
  {
    printf("Impossible d'allouer le tableau vec\ n");
    exit(1);
  }

if ((x = (double *)calloc(N + 1, sizeof(double))) == NULL)
  {
    printf("Impossible d'allouer le tableau x\ n");
    exit(1);
  }

A = (double **)calloc(N + 1, sizeof(double *));
for (i = 0; i < N + 1; i++)
  {
    if ((A[i] = (double *)calloc(7, sizeof(double))) == NULL)
      {
        printf("Impossible d'allouer le tableau A\ n");
        exit(1);
      }
  }


/* résolution du systeme linéaire mx=S par GMRES */

/* précision de la solution */

erreur = 0.001;



/* factorisation incomplete LU : ILU(0) stocké dans A sans la diago de 1 de L

/*Memory Allocation*/




for (i = 0; i < N; i++)
  {
```

```
    for (j = 0; j < 7; j++)
      {

        A[i][j] = m[i][j];

      }
  }



for (i = 1; i < ndr; i++)
  {
    I1 = i - 1;


    A[i][1] /= A[I1][0];


    A[i][0] -= A[i][1] * A[I1][2];

    A[i][4] -= A[i][1] * A[I1][6];



  }






i = ndr;
I1 = i - ndr;
I3 = i - 1;



A[i][3] /= A[I1][0];
```

```
A[i][0] -= A[i][3] * A[I1][4];
A[i][2] -= A[i][3] * A[I1][6];


A[i][1] /= A[I3][0];


A[i][0] -= A[i][1] * A[I3][2];

A[i][4] -= A[i][1] * A[I3][6];



for (i = ndr + 1; i < N - ndr - 1; i++)
  {

    I1 = i - ndr - 1;
    I2 = I1 + 1;
    I4 = i - 1;

    A[i][5] /= A[I1][0];

    A[i][3] -= A[i][5] * A[I1][2];
    A[i][1] -= A[i][5] * A[I1][4];
    A[i][0] -= A[i][5] * A[I1][6];





    A[i][3] /= A[I2][0];




    A[i][0] -= A[i][3] * A[I2][4];
    A[i][2] -= A[i][3] * A[I2][6];
```

```
    A[i][1] /= A[I4][0];


    A[i][0] -= A[i][1] * A[I4][2];

    A[i][4] -= A[i][1] * A[I4][6];



  }

i = N - ndr - 1;

I4 = i - ndr - 1;
I3 = I4 + 1;
I2 = i - 1;

A[i][5] /= A[I4][0];

A[i][3] -= A[i][5] * A[I4][2];
A[i][1] -= A[i][5] * A[I4][4];
A[i][0] -= A[i][5] * A[I4][6];





A[i][3] /= A[I3][0];



A[i][0] -= A[i][3] * A[I3][4];
A[i][2] -= A[i][3] * A[I3][6];




A[i][1] /= A[I2][0];


A[i][0] -= A[i][1] * A[I2][2];
```

```
A[i][4] -= A[i][1] * A[I2][6];



for (i = N - ndr; i < N - 1; i++)
  {

    I1 = i - ndr - 1;
    I2 = I1 + 1;

    I4 = i - 1;


    A[i][5] /= A[I1][0];

    A[i][3] -= A[i][5] * A[I1][2];
    A[i][1] -= A[i][5] * A[I1][4];
    A[i][0] -= A[i][5] * A[I1][6];



    A[i][3] /= A[I2][0];



    A[i][0] -= A[i][3] * A[I2][4];
    A[i][2] -= A[i][3] * A[I2][6];



    A[i][1] /= A[I4][0];


    A[i][0] -= A[i][1] * A[I4][2];


  }


i = N - 1;
```

```
I1 = i - ndr - 1;
I2 = I1 + 1;

I4 = i - 1;


A[i][5] /= A[I1][0];

A[i][3] -= A[i][5] * A[I1][2];
A[i][1] -= A[i][5] * A[I1][4];
A[i][0] -= A[i][5] * A[I1][6];



A[i][3] /= A[I2][0];



A[i][0] -= A[i][3] * A[I2][4];



A[i][1] /= A[I4][0];


A[i][0] -= A[i][1] * A[I4][2];

for (i = 0; i < N; i++)
  {

    /*
      x[i]=Pn[i];
    */
    /* LAGRANGE INTERPOLATION  de degré 3 */
    /*
      x[i]=Pnnn[i]+3*(Pn[i]-Pnn[i]);
    */
    /* LAGRANGE INTERPOLATION  de degré 2 */

    x[i] = -Pnn[i] + 2 * Pn[i];
```

```
  }

/* stockage des solutions  */

for (i = 0; i < N; i++)
  {


    Pnnn[i] = Pnn[i];
    Pnn[i] = Pn[i];

  }


/* initialisation  */

for (i = 0; i < N; i++)
  {
    aux[i] = 0;
  }


/* matrice creuse */
for (i = 1; i < ndr - 1; i++)
  {
    for (j = 1; j < ndr - 1; j++)
      {

        I = i * ndr + j;
        I1 = I + ndr;
        I2 = I - ndr;

        aux[I] = s[I] - m[I][5] * x[I2 - 1] - m[I][3] * x[I2] - m[I][1] * x[I
                 - m[I][4] * x[I1] - m[I][6] * x[I1 + 1];
      }
  }

for (i = 1; i < ndr - 1; i++)
  {
    J = (ndr - 1) * ndr + i;
```

```
    I1 = i + ndr;
    I2 = J - ndr;


    aux[i] = s[i] - m[i][1] * x[i - 1] - m[i][0] * x[i] - m[i][2] * x[i + 1] -
    aux[J] = s[J] - m[J][5] * x[I2 - 1] - m[J][3] * x[I2] - m[J][1] * x[J - 1]

  }

for (i = 1; i < ndr - 1; i++)
  {
    I = i * ndr;
    I1 = I + ndr;
    I2 = I - ndr;


    aux[I] = s[I] - m[I][3] * x[I2] - m[I][0] * x[I] - m[I][2] * x[I + 1] -
             m[I][4] * x[I1] - m[I][6] * x[I1 + 1];

    I = i * ndr + ndr - 1;
    I1 = I + ndr;
    I2 = I - ndr;

    aux[I] = s[I] - m[I][5] * x[I2 - 1] - m[I][3] * x[I2] - m[I][1] * x[I - 1]
             m[I][4] * x[I1];

  }

I = (ndr - 1) * ndr + ndr - 1;
I1 = I - ndr;


aux[I] = s[I] - m[I][5] * x[I1 - 1] - m[I][3] * x[I1] - m[I][1] * x[I - 1] - m

I = (ndr - 1) * ndr;
I1 = I - ndr;

aux[I] = s[I] - m[I][3] * x[I1] - m[I][0] * x[I] - m[I][2] * x[I + 1];
I = 0 + 0;

aux[I] = s[I] - m[I][0] * x[I] - m[I][2] * x[I + 1] - m[I][4] * x[I + ndr] - m
```

```
I = 0 + ndr - 1;

aux[I] = s[I] - m[I][1] * x[I - 1] - m[I][0] * x[I] - m[I][4] * x[I + ndr];


soLU(ndr, A, aux);




som = 0;


for (i = 0; i < N; i++)
  {

    som += aux[i] * aux[i];
  }

res = som;



res = sqrt(res);


if (res <= erreur)
  {

    for (i = 0; i < N; i++)
      {
        Pn[i] = x[i];
      }


    return 0;
  }

/* demarrage des iterations */
```

```
for (it = 0; it < nit; it++)
  {


    nk = nkr;

    /* orthonormalisation d'Arnoldi */



    for (i = 0; i < N; i++)
      {
        aux[i] /= res;

      }
    for (i = 1; i < nkrMax; i++)
      {
        rr[i] = 0;
      }
    rr[0]   = res ;

    for (j = 0; j < nkr; j++)
      {

        for (i = 0; i < N; i++)
          {
            v[i][j] = aux[i];
          }


        /* matrice creuse */



        for (i = 1; i < ndr - 1; i++)
          {
            for (k = 1; k < ndr - 1; k++)
              {

                I = i * ndr + k;
```

```
          I1 = I + ndr;
          I2 = I - ndr;

          aux1[I] = m[I][5] * aux[I2 - 1] + m[I][3] * aux[I2] + m[I][1]
                    m[I][4] * aux[I1] + m[I][6] * aux[I1 + 1];

        }
    }



for (i = 1; i < ndr - 1; i++)
  {
    J = (ndr - 1) * ndr + i;

    aux1[i] = m[i][1] * aux[i - 1] + m[i][0] * aux[i] + m[i][2] * aux[
    aux1[J] = m[J][5] * aux[J - ndr - 1] + m[J][3] * aux[J - ndr] + m[

  }



for (i = 1; i < ndr - 1; i++)
  {
    I = i * ndr;

    aux1[I] = m[I][3] * aux[I - ndr] + m[I][0] * aux[I] + m[I][2] * au

    I = i * ndr + ndr - 1;

    aux1[I] = m[I][5] * aux[I - ndr - 1] + m[I][3] * aux[I - ndr] + m[

  }

I = (ndr - 1) * ndr + ndr - 1;

aux1[I] = m[I][5] * aux[I - ndr - 1] + m[I][3] * aux[I - ndr] + m[I][1

I = (ndr - 1) * ndr;

aux1[I] = m[I][3] * aux[I - ndr] + m[I][0] * aux[I] + m[I][2] * aux[I
```

```
I = 0 + 0;

aux1[I] = m[I][0] * aux[I] + m[I][2] * aux[I + 1] + m[I][4] * aux[I +

I = 0 + ndr - 1;

aux1[I] = m[I][1] * aux[I - 1] + m[I][0] * aux[I] + m[I][4] * aux[I +



soLU(ndr, A, aux1);

for (i = 0; i < N; i++)
  {
    aux[i] = aux1[i];
  }


for (i = 0; i < j + 1; i++)
  {
    for (k = 0; k < N; k++)
      {
        vec[k] = v[k][i];
      }


    som = 0;
    for (i1 = 0; i1 < N; i1++)
      {
        som += aux1[i1] * vec[i1];
      }
    tem = som;
    hh[i][j] = tem;
    for (i1 = 0; i1 < N; i1++)
      {
        aux[i1] -= tem * v[i1][i];
      }
  }
```

```
    som = 0;
    for (i1 = 0; i1 < N; i1++)
      {
        som += aux[i1] * aux[i1];
      }


    dem = som;
    dem = sqrt(dem);
    hh[j + 1][j] = dem;

    if (dem <= erreur)
      {


        nk = j;
        for (i = 0; i < N; i++)
          {
            v[i][j + 1] = aux[i];
          }

        break;

      }
    else
      {
        for (i = 0; i < N; i++)
          {
            aux[i] /= dem;
          }
      }
  }

/* triangularisation et modif. du second membre */


for (i = 0; i < nk; i++)
  {
    ip = i + 1;
    hii = hh[i][i];
```

```
    hipi = hh[ip][i];
    gamm = hii * hii + hipi * hipi;
    gamm = sqrt(gamm);
    gamm = 1. / gamm;
    coss = hii * gamm;
    sinn = -hipi * gamm;

    for (j = i; j < nk; j++)
      {
        hij = hh[i][j];
        hipj = hh[ip][j];
        hh[i][j] = coss * hij - sinn * hipj;
        hh[ip][j] = sinn * hij + coss * hipj;
      }
    raux = rr[i];
    rr[i] = coss * raux;
    rr[ip] = sinn * raux;
  }


/* resolution du systeme triangulaire superieur */
for (i1 = nk - 1; i1 > -1; i1--)
  {
    tem = rr[i1] / hh[i1][i1];
    rr[i1] = tem;
    for (j11 = i1 - 1; j11 > -1; j11--)
      {

        rr[j11] -= hh[j11][i1] * tem;
      }
  }

for (i1 = 0; i1 < nk; i1++)
  {
    tem = rr[i1];
    for (j11 = 0; j11 < N; j11++)
      {
        x[j11] += tem * v[j11][i1];
      }
  }
```

```
/* calcul du residu */

/* matrice creuse  */


for (i = 1; i < ndr - 1; i++)
  {
    for (j = 1; j < ndr - 1; j++)
      {

        I = i * ndr + j;
        I1 = I + ndr;
        I2 = I - ndr;

        aux[I] = s[I] - m[I][5] * x[I2 - 1] - m[I][3] * x[I2] - m[I][1] *
                 m[I][4] * x[I1] - m[I][6] * x[I1 + 1];


      }
  }



for (i = 1; i < ndr - 1; i++)
  {
    J = (ndr - 1) * ndr + i;

    aux[i] = s[i] - m[i][1] * x[i - 1] - m[i][0] * x[i] - m[i][2] * x[i +
    aux[J] = s[J] - m[J][5] * x[J - ndr - 1] - m[J][3] * x[J - ndr] - m[J]

  }

for (i = 1; i < ndr - 1; i++)
  {
    I = i * ndr;

    aux[I] = s[I] - m[I][3] * x[I - ndr] - m[I][0] * x[I] - m[I][2] * x[I
             m[I][4] * x[I + ndr] - m[I][6] * x[I + ndr + 1];

    I = i * ndr + ndr - 1;
```

```
    aux[I] = s[I] - m[I][5] * x[I - ndr - 1] - m[I][3] * x[I - ndr] - m[I]
             m[I][4] * x[I + ndr];

  }

I = (ndr - 1) * ndr + ndr - 1;

aux[I] = s[I] - m[I][5] * x[I - ndr - 1] - m[I][3] * x[I - ndr] - m[I][1]

I = (ndr - 1) * ndr;

aux[I] = s[I] - m[I][3] * x[I - ndr] - m[I][0] * x[I] - m[I][2] * x[I + 1]

I = 0 + 0;

aux[I] = s[I] - m[I][0] * x[I] - m[I][2] * x[I + 1] - m[I][4] * x[I + ndr]

I = 0 + ndr - 1;

aux[I] = s[I] - m[I][1] * x[I - 1] - m[I][0] * x[I] - m[I][4] * x[I + ndr]


soLU(ndr, A, aux);



som = 0;
for (i1 = 0; i1 < N; i1++)
  {
    som += aux[i1] * aux[i1];
  }
res = som;

res = sqrt(res);

if (res <= erreur)
  {

    break;
  }
```

```
      iterr += 1;

    }

  for (i = 0; i < N; i++)
    Pn[i] = x[i];

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

  for (i = 0; i < nkrMax + 1; i++)
    free(hh[i]);
  free(hh);

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


  free(vec);
  free(rr);
  free(aux);
  free(aux1);
  free(x);

  return 1;


}

/**************************************************************************/
static int optionbond_implicit1d(double maturity_option, NumFunc_1 *p, int am,
                                 double t,   /*                  */
                                 double maturity_bond,   /* maturité du zéro-cou
                                 /*                                              *
                                 double alpha0,      /* Paramètres de la volatil
                                 double alphar,      /*
                                 double alphaf,
```

```
                                /*(t,T,r,f) = (alpha0+alphar*r+alphaf*f)^gamma*
                                double gamm0,       /*
                                double lambda,       /*
     */
                                double beta0,          /* Paramètres taux forwa
                                double beta1,         /*
                                double eta,           /* f(0,t) = beta_0 + beta
                                /*
                                double tau,
                                int ndr,        /* nombre de pas de d'espace et
                                int ndf,

                                int ndt,
                                double *price)
{
  int i, j, k, I, ii, kk;
  double temps;
  double *s, **m, **sigmarr;
  double R, F;    /* localisation */
  double dt, dr, df, df2, dr2, drdf, idr, jdf, sigr2, sigf2, muff, murr, sigrf;
  double c0, c1, p1; /* prix interpolé */
  int N;
  double r00 = beta0;   /* (r00,f00)          */
  double f00 = beta0;   /* à l'instant t */

  /* constantes */
  if (tau > maturity_bond)
    return PREMIA_UNTREATED_TAU_BHAR_CHIARELLA;

  N = ndr * ndr;

  /* Localization */
  R = 1;
  F = 1;

  /* steps  */
  dr = R / ndr;
  df = F / ndf;
  dr2 = dr * dr;
  df2 = df * df;
```

```
sigmarr = (double **)calloc(ndr, sizeof(double *));
for (i = 0; i < ndr; i++)
  {
    if ((sigmarr[i] = (double *)calloc(ndf, sizeof(double))) == NULL)
      {
        printf("Impossible d'allouer le tableau sigmarr\ n");
        exit(1);
      }
  }

/* calcul de sigmarr */
for (i = 0; i < ndr; i++)
  {
    idr = i * dr;
    for (j = 0; j < ndf; j++)
      {
        sigmarr[i][j] = exp(gamm0 * log(alpha0 + alphar * idr + alphaf * j * d
      }
  }

m = (double **)calloc(N, sizeof(double *));
for (i = 0; i < N; i++)
  {
    if ((m[i] = (double *)calloc(7, sizeof(double))) == NULL)
      {
        printf("Impossible d'allouer le tableau m\ n");
        exit(1);
      }
  }

if ((s = (double *)calloc(N, sizeof(double))) == NULL)
  {
    printf("Impossible d'allouer le tableau s\ n");
    exit(1);
  }


if ((Pn = (double *)calloc(N, sizeof(double))) == NULL)
  {
    printf("MEMORY_ALLOCATION_FAILURE\ n");
    exit(1);
```

```
  }

if ((Pnn = (double *)calloc(N, sizeof(double))) == NULL)
  {
    printf("MEMORY_ALLOCATION_FAILURE\ n");
    exit(1);
  }
if ((Pnnn = (double *)calloc(N, sizeof(double))) == NULL)
  {
    printf("MEMORY_ALLOCATION_FAILURE\ n");
    exit(1);
  }

for (kk = 0; kk < 2; kk++)
  {
    /* Initialisation */

    if (kk == 0)
      {

        for (i = 0; i < ndr; i++)
          {
            for (j = 0; j < ndf; j++)
              {
                I = i * ndr + j;
                /* bond-pricing */

                Pn[I] = 1;
                Pnn[I] = 1;
                Pnnn[I] = 1;
              }
          }


        dt = (maturity_bond - maturity_option) / ndt;

        drdf = dt * 0.5 / (dr * df);

        temps = maturity_bond;

      }
```

```
else
  {


    /* option pricing */

    for (i = 0; i < ndr; i++)
      {
        for (j = 0; j < ndf; j++)
          {

            I = i * (ndr) + j;

            Pn[I] = (p->Compute)(p->Par, Pn[I]);
            Pnn[I] = Pn[I];
            Pnnn[I] = Pn[I];
          }
      }


    dt = (maturity_option - t) / ndt;

    drdf = dt * 0.5 / (dr * df);

    temps = maturity_option;
  }

/* mise du systeme lineaire a 0 */
for (i = 0; i < ndr; i++)
  {
    for (j = 0; j < ndf; j++)
      {
        I = i * (ndr) + j;
        s[I] = 0;
        for (k = 0; k < 7; k++)
          {
            m[I][k] = 0;
          }
      }
  }
```

```
/* conditions de Neumann homogenes sur le bord valables a chaque pas de te
for (i = 1; i < ndr - 1; i++)
  {
    I = i * ndr + 0;


    m[I][0] = 1;

    m[I][2] = -1;


    I = i * ndr + ndf - 1;

    m[I][0] = 1;


    m[I][1] = -1;
  }

for (j = 1; j < ndf - 1; j++)
  {

    I = 0 + j;

    m[I][0] = 1;

    m[I][4] = -1;


    I = (ndr - 1) * ndr + j;

    m[I][0] = 1;

    m[I][3] = -1;

  }

/* Les 4 angles */
```

```
I = (ndr - 1) * ndr + ndf - 1;

m[I][0] = 1;

m[I][5] = -1;

I = (ndr - 1) * ndr;

m[I][0] = 1;


m[I][2] = -0.5;
m[I][3] = -0.5;
I = 0 + 0;

m[I][0] = 1;



m[I][6] = -1;



I = 0 + ndf - 1;


m[I][0] = 1;

m[I][1] = -0.5;
m[I][4] = -0.5;

for (ii = 0; ii < ndt; ii++)
  {

    temps -= dt;

    /* remplissage du systeme lineaire */

    for (i = 1; i < ndr - 1; i++)
      {
        idr = i * dr;
```

```
        for (j = 1; j < ndf - 1; j++)
          {
            jdf = j * df;

            sigr2 = sigmarr[i][j];

            sigf2 = sigma1(temps, idr, jdf, lambda, tau, gamm0, alpha0, al

            sigrf = sigr2 * sigf2 * drdf;

            sigf2 = sigf2 * sigf2 * dt / df2;

            sigr2 = sigr2 * sigr2 * dt / dr2;

            murr = mur(temps, idr, jdf, lambda, beta0, beta1, eta, tau) *

            muff = mu1(temps, idr, jdf, tau, lambda, gamm0, alpha0, alphar

            I = i * ndr + j;
            m[I][0] = 1 + sigf2 + sigr2 + idr * dt - 2 * sigrf;

            m[I][1] =  muff - sigf2 * 0.5 + sigrf ;
            m[I][2] = m[I][1] - 2 * muff;
            m[I][3] = murr - sigr2 * 0.5 + sigrf;
            m[I][4] = -murr - sigr2 * 0.5 + sigrf;
            m[I][5] = -sigrf;
            m[I][6] = -sigrf;
            s[I] = Pn[I];

          }
      }


    /* resolution du systeme lineaire */
    resolution(ndr, m, Pn, Pnn, Pnnn, s);

  }

/* Interpolation bilineaire pour le prix */
i = 0;
while (r00 > i * dr)
```

```
        i++;
      j = 0;
      while (f00 > j * df)
        j++;

      c0 = (r00 - (i - 1) * dr) / dr;
      c1 = (f00 - (j - 1) * df) / df;


      p1 = (1. - c0) * (1. - c1) * Pn[(i - 1) * ndr + j - 1] + c0 * (1. - c1) *
           c0 * c1 * Pn[i * ndr + j] + (1. - c0) * c1 * Pn[(i - 1) * ndr + j];
    }

  /*Price*/
  *price = p1;

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

  for (i = 0; i < ndr; i++)
    free(sigmarr[i]);
  free(sigmarr);

  free(Pn);
  free(Pnn);
  free(Pnnn);
  free(s);

  return OK;
}

int CALC(FD_IMPLICIT_ZBO)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  return optionbond_implicit1d(ptOpt->OMaturity.Val.V_DATE, ptOpt->PayOff.Val.V_
}
```

```
static int CHK_OPT(FD_IMPLICIT_ZBO)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "ZeroCouponCallBondEuro") == 0) | (strcmp(((
    return OK;
  else
    return WRONG;
}

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

      Met->Par[0].Val.V_LONG = 100;
      Met->Par[1].Val.V_LONG = 100;
      Met->Par[2].Val.V_LONG = 100;
    }
  return OK;
}

PricingMethod MET(FD_IMPLICIT_ZBO) =
{
  "FD_Implicit_BharChiarella1d_ZBO",
  { {"TimeStepNumber", LONG, {100}, ALLOW}, {"SpotRateSpaceStepNumber", LONG, {1
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(FD_IMPLICIT_ZBO),
  {{"Price", DOUBLE, {100}, FORBID} , {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(FD_IMPLICIT_ZBO),
  CHK_ok,
  MET(Init)
} ;
```