

[Help](#)

```
#include "bharchiarella1d_std.h"

static int resolution05(int taille_trid, double **Aj, double *Bj, double *xj)
{
    int i;

    /* résolution du systeme linéaire  $Aj*xj=Bj$  pivot gauss */

    for (i = 0; i < taille_trid - 1; i++)
    {
        Aj[2][i] = Aj[2][i] / Aj[1][i];

        Bj[i] = Bj[i] / Aj[1][i];

        Aj[1][i] = 1;

        /*
            for(k=i+1;k<i+2;k++){
        */
        Aj[1][i + 1] = Aj[1][i + 1] - Aj[0][i + 1] * Aj[2][i];

        Bj[i + 1] = Bj[i + 1] - Aj[0][i + 1] * Bj[i];
        /*
            }
        */
    }

    Bj[taille_trid - 1] = Bj[taille_trid - 1] / Aj[1][taille_trid - 1];

    Aj[1][taille_trid - 1] = 1;

    /* resolution systeme triangulaire superieur */

    /* Resoudre  $UX=Y$  par remontée triangulaire */
}
```

```
xj[taille_trid - 1] = Bj[taille_trid - 1];

for (i = taille_trid - 2; i > -1; i--)
{
    xj[i] = Bj[i];

    xj[i] -= Aj[2][i] * xj[i + 1];

}

return 0;
}

static double f0(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, double beta0, double beta1)
{
    if (t > 0)
    {
        if (t < tau)
        {
            return (

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

```

```

/*****/

static double mur(double t, double x, double y, double lambda, double beta0, double beta1, double eta)
{
    return (D(t, beta0, beta1, eta, lambda) + psi(t, x, y, lambda, tau, beta0, beta1));
}

/*****/

static double sigmar(double x, double y, double gamma0, double alpha0, double alphas, double alphaf)
{
    return (exp(gamma0 * log(alpha0 + alphas * x + alphaf * y)));
}

/*****/

static double sigma1(double t, double x, double y, double lambda, double tau, double gamma0, double alpha0, double alphas, double alphaf)
{
    return (exp(-lambda * tau + lambda * t) * sigmar(x, y, gamma0, alpha0, alphas, alphaf));
}

/*****/

static double mu1(double t, double x, double y, double tau, double lambda, double gamma0, double alpha0, double alphas, double alphaf)
{
    return (sigma1(t, x, y, lambda, tau, gamma0, alpha0, alphas, 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))));
*
* } */

*****/

static int optionbond_adild(double maturity_option, NumFunc_1 *p, int am,
                           double t, /* */
                           double maturity_bond, /* maturité du zéro-coupon */
                           /* */
                           double alpha0, /* Paramètres de la volatilité
                           double alphas, /*
                           double alphaf,

                           /*(t,T,r,f) = (alpha0+alphas*r+alphaf*f)^gamma*exp(-
                           double gamma0, /*
                           double lambda, /*

                           */

                           double beta0, /* Paramètres taux forward
                           double beta1, /*
                           double eta, /* f(0,t) = beta_0 + beta_1*(1
                           /*
                           double tau,
                           int ndr, /* nombre de pas de d'espace et de t
                           int ndf,

                           int ndt,
                           double *price)

{
  int i, j, I, k, kk;
  double temps;
  double df2, dr2, drdf, idr, jdf, sigr2, sigf2, sigrf, theta, theta1;
  double dt, dr, df;
  int N; /* Taille des systèmes */

```

```

double R = 1., F = 1.; /* Localisation spatiale */
double **sigmarr;
double c0, c1, murr, muff;
double *Pn, *Pn05; /* Vecteurs des prix sur la grille (r,f) */
double **Aj, *Bj, *xj;

double **Ai, *Bi, *xi;
double r00 = beta0; /* (r00,f00) */
double f00 = beta0; /* à l'instant t */

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

theta = 12;
theta1 = 1 / theta;
N = ndr * ndf;

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

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

if ((Pn05 = (double *)calloc(N + 1, sizeof(double))) == NULL)
{
    printf("Impossible d'allouer le tableau Pn05\ n");
    exit(1);
}

Aj = (double **)calloc(3, sizeof(double *));
for (i = 0; i < 3; i++)
{
    if ((Aj[i] = (double *)calloc(ndr, sizeof(double))) == NULL)

```

```
        {
            printf("Impossible d'allouer le tableau Aj\ n");
            exit(1);
        }
    }

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

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

    Ai = (double **)calloc(3, sizeof(double *));
    for (i = 0; i < 3; i++)
    {
        if ((Ai[i] = (double *)calloc(ndf, sizeof(double))) == NULL)
        {
            printf("Impossible d'allouer le tableau Ai\ n");
            exit(1);
        }
    }

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

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

    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);
    }
}

/* sigmarr */
for (i = 0; i < ndr; i++)
{
    idr = i * dr;
    for (j = 0; j < ndf; j++)
    {
        sigmarr[i][j] = exp(gamma0 * log(alpha0 + alphas * idr + alphaf * j *
    }
}

for (kk = 0; kk < 2; kk++)
{
    if (kk == 0)
    {
        /* PAYOFF Computation*/

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

                I = i * ndr + j;

                /* bond-pricing */

                Pn[I] = 1;
                Pn05[I] = 0;
            }
        }
    }
}

```



```

    temps = maturity_bond;
    dt = (maturity_bond - maturity_option) / ndt;

    drdf = dt * 0.25 / (dr * df);
}
else
{
    for (i = 0; i < ndr; i++)
    {
        for (j = 0; j < ndf; j++)
        {

            I = i * ndr + j;
            /* option-pricing */
            Pn[I] = (p->Compute)(p->Par, Pn[I]);
            Pn05[I] = 0;

        }
    }
    temps = maturity_option;
    dt = (maturity_option - t) / ndt;

    drdf = dt * 0.25 / (dr * df);
}

/* Initialization of matrix elements */

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

    Aj[0][i] = 0;
    Aj[1][i] = 0;
    Aj[2][i] = 0;
    Bj[i] = 0;

}

for (j = 0; j < ndf; j++)
{
    Ai[0][j] = 0;

```

```

    Ai[1][j] = 0;
    Ai[2][j] = 0;
    Bi[j] = 0;
}

for (k = 0; k < ndt; k++)
{
    temps -= dt;

    /* First Direction */

    for (j = 1; j < ndf - 1; j++)
    {
        jdf = j * df;
        for (i = 1; i < ndr - 1; i++)
        {
            I = i * ndr + j;
            idr = i * dr;

            sigr2 = sigmarr[i][j];

            sigf2 = sigma1(temps + dt, idr, jdf, lambda, tau, gamma0, alph
            sigrf = sigr2 * sigf2 * drdf;
            sigf2 = sigf2 * sigf2 * dt / df2;
            sigr2 = 0.25 * sigr2 * sigr2 * dt / dr2;
            murr = mur(temps + 0.5 * dt, idr, jdf, lambda, beta0, beta1, e

            Aj[0][i] = ((murr - sigr2) + theta1);

            Aj[1][i] = 1 - 2 * (theta1 - sigr2);

            Aj[2][i] = -(murr + sigr2) + theta1);

            Bj[i] = Pn[I] * (1 - (idr * dt + 2 * sigr2 + sigf2) - 2 * thet
                (Pn[I + 1] + Pn[I - 1]) * 0.5 * sigf2 + sigrf * (Pn[I

```

```
    }

    /* Neumann Boundary Conditions */

    i = 0;

    Aj[0][i] = 0;
    Aj[1][i] = 1;
    Aj[2][i] = -1;

    Bj[i] = 0;

    i = ndr - 1;

    Aj[0][i] = -1;
    Aj[1][i] = 1;
    Aj[2][i] = 0;

    Bj[i] = 0;

    /* Solve linear system */

    resolution05(ndr, Aj, Bj, xj);

    for (i = 0; i < ndr; i++)
    {
        I = i * ndr + j;
        Pn05[I] = xj[i];
    }

}

/* Neumann Boundary Conditions */

j = 0;
for (i = 1; i < ndr - 1; i++)
{
    I = i * ndr + j;
```

```

        Pn05[I] = Pn05[I + 1];
    }
    Pn05[0] = Pn05[ndr + 1];

    Pn05[(ndr - 1)*ndr] = Pn05[(ndr - 2) * ndr + 1];

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

        Pn05[I] = Pn05[I - 1];

    }

    Pn05[ndf - 1] = Pn05[ndr + ndf - 2];
    Pn05[(ndr - 1)*ndr + ndf - 1] = Pn05[(ndr - 2) * ndr + ndf - 2];


/* Second Direction */

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

    idr = i * dr;

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

        I = i * ndr + j;

        jdf = j * df;

        sigf2 = sigma1(temps - dt, idr, jdf, lambda, tau, gamma0, alpha0, alpha1);

        sigf2 = 0.25 * sigf2 * sigf2 * dt / df2;

        muff = mu1(temps, idr, jdf, tau, lambda, gamma0, alpha0, alpha1, alpha2);
    }
}

```

```

    Ai[0][j] = (muff - sigf2) + theta1;

    Ai[1][j] = 1 - 2 * (-sigf2 + theta1);

    Ai[2][j] = -(muff + sigf2) + theta1;

    Bi[j] = Pn05[I] + (theta1 - sigf2) * (Pn[I + 1] - 2 * Pn[I] +

}

/* Neumann Boundary Conditions */
j = 0;

Ai[0][j] = 0;
Ai[1][j] = 1;
Ai[2][j] = -1;

Bi[j] = 0;

j = ndf - 1;

Ai[0][j] = -1;
Ai[1][j] = 1;
Ai[2][j] = 0;

Bi[j] = 0;

/* Solve linear system */

resolution05(ndf, Ai, Bi, xi);

for (j = 0; j < ndf; j++)
{
    I = i * ndr + j;
    Pn[I] = xi[j];
}
}

```

```

/* Neumann Boundary Conditions */

i = 0;
for (j = 1; j < ndf - 1; j++)
{
    I = i * ndr + j;
    Pn[I] = Pn[I + ndr];
}
Pn[0] = Pn[ndr + 1];

Pn[ndf - 1] = Pn[ndr * ndr + ndf - 2];

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

    Pn[I] = Pn[I - ndr];

}

Pn[(ndr - 1)*ndr] = Pn[(ndr - 2) * ndr + 1];
Pn[(ndr - 1)*ndr + ndf - 1] = Pn[(ndr - 2) * ndr + ndf - 2];
}

/* Bilinear Interpolation */

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;
}

/*Price*/
*price = ((1. - c0) * (1. - c1) * Pn[(i - 1) * ndr + j - 1] + c0 * (1. - c1) *

```

```

    for (i = 0; i < 3; i++)
        free(Ai[i]);
    free(Ai);

    for (i = 0; i < 3; i++)
        free(Aj[i]);
    free(Aj);

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

    free(xj);
    free(xi);
    free(Pn);
    free(Pn05);
    free(Bj);
    free(Bi);

    return OK;
}

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

    return optionbond_adi1d(ptOpt->OMaturity.Val.V_DATE, ptOpt->PayOff.Val.V_NUMFU
}

static int CHK_OPT(FD_ADI_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_ADI_ZBO) =
{
    "FD_Adi_BharChiarella1d_ZBO",
    { {"TimeStepNumber", LONG, {100}, ALLOW}, {"SpotRateSpaceStepNumber", LONG, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID} },
    CALC(FD_ADI_ZBO),
    { {"Price", DOUBLE, {100}, FORBID} , {" ", PREMIA_NULLTYPE, {0}, FORBID} },
    CHK_OPT(FD_ADI_ZBO),
    CHK_ok,
    MET(Init)
} ;

```