

## [Help](#)

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

/*Computation of the grid*/
static void grid(double So, double sigma, double r, double d, double K, double h)
{
    int i;
    double alpha;
    double avt = 0, ap = 0, longu = 0;
    double test;
    double mu = r - sigma * sigma / 2 - d;
    double rho = exp(-r * h);

    /*The abscisses*/
    a[0] = 0;
    a[1] = exp(mu * t[n - 1] - 3 * sigma * sqrt(t[n - 1]));
    a[p - 1] = exp(mu * t[n - 1] + 3 * sigma * sqrt(t[n - 1]));
    a[p] = exp(mu * t[n - 1] + 4 * sigma * sqrt(t[n - 1]));
    for (i = 2; i <= p - 2; i++)
    {
        alpha = ((double)i - 1) / ((double)p - 2);
        avt = a[i - 1];
        ap = a[p - 1];
        test = 0;
        a[i] = (avt + ap) / 2;
        test = cdf_nor((log(a[i]) - mu * t[n - 1]) / (sigma * sqrt(t[n - 1]))) - a
        while (fabs(test) > 0.0000001)
        {
            if (test >= 0)
            {
                ap = a[i];
                a[i] = (avt + ap) / 2;
            }
            else
            {
                avt = a[i];
                a[i] = (avt + ap) / 2;
            }
        }
    }
}
```

```

        test = cdf_nor((log(a[i]) - mu * t[n - 1]) / (sigma * sqrt(t[n - 1])))
    }
}
for (i = 1; i <= p; i++)
{
    a[i] = So * a[i];
}

/*The ordonnees*/
b[0] = 0;
b[1] = So * exp(mu * t[n - 1] - 2 * sigma * sqrt(t[n - 1]));
b[q / 4] = ((n - 1) * rho - 1) * K / (n - 2);
b[3 * q / 4] = n * K / (n - 2);
b[q] = So * exp(mu * t[n - 1] + 3.9 * sigma * sqrt(t[n - 1]));
for (i = 2; i <= q / 4 - 1; i++)
{
    longu = b[q / 4] - b[1];
    b[i] = b[1] + longu * (i - 1) / (q / 4 - 1);
}
for (i = q / 4 + 1; i <= 3 * q / 4 - 1; i++)
{
    longu = b[3 * q / 4] - b[q / 4];
    b[i] = b[q / 4] + longu * (i - q / 4) / (3 * q / 4 - q / 4);
}
for (i = 3 * q / 4 + 1; i <= q - 1; i++)
{
    longu = b[q] - b[3 * q / 4];
    b[i] = b[3 * q / 4] + longu * (i - 3 * q / 4) / (q - 3 * q / 4);
}

}

/*Computation of the Pik Qik*/
static void PQ(double So, double sigma, double r, double d, double K, double h,
{
    int i, k;
    double c1 = h * (r - sigma * sigma / 2 - d);
    double c2 = sigma * sqrt(h);

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

```

```

        for (k = 1; k <= p; k++)
        {
            P[i][k] = cdf_nor((log(a[i + 1] / a[k]) - c1) / c2) - cdf_nor((log(a[i] / a[k]) - c1) / c2);
            Q[i][k] = exp(c2 * c2 / 2 + c1) * (cdf_nor((log(a[i + 1] / a[k]) - c1) / c2) - cdf_nor((log(a[i] / a[k]) - c1) / c2));
        }
    }
P[0][0] = 1 - cdf_nor(-c1 / c2);
Q[0][0] = exp(c2 * c2 / 2 + c1) * (1 - cdf_nor(-c1 / c2 - c2));;

for (i = 1; i <= p; i++)
{
    P[i][0] = 0;
    Q[i][0] = 0;
}
for (k = 1; k <= p; k++)
{
    P[0][k] = cdf_nor((log(a[1] / a[k]) - c1) / c2);
    Q[0][k] = exp(c2 * c2 / 2 + c1) * cdf_nor((log(a[1] / a[k]) - c1) / c2 - c2);
}
for (k = 1; k <= p; k++)
{
    P[p][k] = 1 - cdf_nor((log(a[p] / a[k]) - c1) / c2);
    Q[p][k] = exp(c2 * c2 / 2 + c1) * (1 - cdf_nor((log(a[p] / a[k]) - c1) / c2 - c2));
}

}

/*To solve the 4x4 system*/
static void solve(int p, int q, double a[], double b[], double ***coeff, double w1, w2, w3, w4)
{
    double ak, akp, bl, blp;
    int k, l;

    for (k = 0; k <= p - 1; k++)
    {
        for (l = 0; l <= q - 1; l++)
        {
            ak = a[k];
            akp = a[k + 1];
            bl = b[l];
            blp = b[l + 1];

```

```

        w1 = points[k][1];
        w2 = points[k + 1][1];
        w3 = points[k][1 + 1];
        w4 = points[k + 1][1 + 1];
        coeff[k][1][3] = (w4 - w3 - w2 + w1) / ((blp - bl) * (akp - ak));
        coeff[k][1][1] = (w2 - w1) / (akp - ak) - coeff[k][1][3] * bl;
        coeff[k][1][2] = (w3 - w1) / (blp - bl) - coeff[k][1][3] * ak;
        coeff[k][1][0] = w1 - coeff[k][1][1] * ak - coeff[k][1][2] * bl - coef
    }
}
for (k = 0; k <= p - 1; k++)
{
    coeff[k][q][0] = coeff[k][q - 1][0];
    coeff[k][q][1] = coeff[k][q - 1][1];
    coeff[k][q][2] = coeff[k][q - 1][2];
    coeff[k][q][3] = coeff[k][q - 1][3];
}
for (l = 0; l <= q - 1; l++)
{
    coeff[p][1][0] = coeff[p - 1][1][0];
    coeff[p][1][1] = coeff[p - 1][1][1];
    coeff[p][1][2] = coeff[p - 1][1][2];
    coeff[p][1][3] = coeff[p - 1][1][3];
}
coeff[p][q][0] = coeff[p - 1][q - 1][0];
coeff[p][q][1] = coeff[p - 1][q - 1][1];
coeff[p][q][2] = coeff[p - 1][q - 1][2];
coeff[p][q][3] = coeff[p - 1][q - 1][3];
}

/*Calculation of wn-1*/
static double vnMoinsUn(double s, double sbprime, double sigma, double r, double
{
    double moy = (n == 1) ? s : ((n - 2) * sbprime + s) / (n - 1);
    /* double payoff=moy-K;*/
    double rho = exp(-r * h);
    double KBarre = n * K - (n - 1) * moy;
    double res = 0;
    double d1 = (log(s / KBarre) + (r - d + sigma * sigma / 2) * h) / (sigma * sqr
    double vBS;

```

```

if (option == 1)
{
    /*case of the amerasian call*/
    if (KBarre <= 0)
    {
        res = MAX(moy - K, (s * exp(-d * h) + (double)(n - 1) * rho * moy) / (
    }
    else
    {
        vBS = (cdf_nor(d1) * s * exp(-d * h) - rho * KBarre * cdf_nor(d1 - sig
        res = MAX(moy - K, vBS);
    }
}
if (option == 3)
{
    /*case of the eurAsian call*/
    if (KBarre <= 0)
    {
        res = (s * exp(-d * h) + (n - 1) * rho * moy) / n - rho * K;
    }
    else
    {
        vBS = (cdf_nor(d1) * s * exp(-d * h) - rho * KBarre * cdf_nor(d1 - sig
        res = vBS;
    }
}
if (option == 2)
{
    /*case of the amerasian put*/
    if (KBarre <= 0)
    {
        res = 0;
    }
    else
    {
        vBS = -(cdf_nor(-d1) * s * exp(-d * h) - rho * KBarre * cdf_nor(-d1 +
        res = MAX(K - moy, vBS);
    }
}

```

```

if (option == 4)
{
    /*case of the eurAsian put*/
    if (KBarre <= 0)
    {
        res = 0;
    }
    else
    {
        vBS = -(cdf_nor(-d1) * s * exp(-d * h) - rho * KBarre * cdf_nor(-d1 +
        res = vBS;
    }
}
return res;
}

```

```

static int Fixed_BenHameurBretonLecuyer(int am_or_eu, NumFunc_2 *Payoff, double
{

```

```

    /*initialization*/
    int i, j, k, l, m, v, ksi, option;
    double cc0, cc2, ckl, p1, p2, delta, super, K, So;
    double h = T / n;
    double *t;
    double rho = exp(-r * h);
    double d1;
    double *a;
    double *b;
    double **P;
    double **Q;
    double ***coeff;
    double **points;

```

```

    t = malloc((n + 1) * sizeof(double));
    a = malloc((p + 1) * sizeof(double));
    b = malloc((q + 1) * sizeof(double));

```

```

    P = malloc((p + 1) * sizeof(double *));
    for (i = 0; i <= p; i++)
        P[i] = malloc((p + 1) * sizeof(double));
    Q = malloc((p + 1) * sizeof(double *));

```

```

for (i = 0; i <= p; i++)
    Q[i] = malloc((p + 1) * sizeof(double));

points = malloc((p + 1) * sizeof(double *));
for (k = 0; k <= p; k++)
    points[k] = malloc((q + 1) * sizeof(double *));

coeff = malloc((p + 1) * sizeof(double **));
for (k = 0; k <= p; k++)
    coeff[k] = malloc((q + 1) * sizeof(double *));
for (k = 0; k <= p; k++)
    for (l = 0; l <= q; l++)
        coeff[k][l] = malloc(4 * sizeof(double));

if (((Payoff->Compute) == &Call_OverSpot2)) && (am_or_eu == 0))
    option = 3;
else if (((Payoff->Compute) == &Call_OverSpot2)) && (am_or_eu == 1))
    option = 1;
else if (((Payoff->Compute) == &Put_OverSpot2)) && (am_or_eu == 0))
    option = 4;
else /*if (((Payoff->Compute) == &Put_OverSpot2)) && (am_or_eu == 1))*/
    option = 2;

K = pseudo_strike;
So = pseudo_stock;
d1 = (log(So / K) + (r - d + sigma * sigma / 2.) * h) / (sigma * sqrt(h));

/*initialization of the dates*/
for (i = 0; i <= n; i++)
{
    t[i] = i * h;
}
/*initialization of the grid*/
grid(So, sigma, r, d, K, h, n, p, q, t, a, b);
/*initialization of the Pik*/
PQ(So, sigma, r, d, K, h, n, p, q, t, a, b, P, Q);

/*computation of the approximated wn-1*/
for (k = 0; k <= p; k++)
{

```

```

        for (l = 0; l <= q; l++)
        {
            points[k][l] = vnMoinsUn(a[k], b[l], sigma, r, d, K, h, n, option);
        }
    }
    solve(p, q, a, b, coeff, points);

/*The recursion*/
for (m = n - 2; 2 <= m; m--)
{
    /*computation of the value at the points*/
    for (k = 0; k <= p; k++)
    {
        for (l = 0; l <= q; l++)
        {
            ckl = (m == 0) ? 0 : ((m - 1) * b[l] + a[k]) / ((double) m);
            v = 0;
            ksi = 0;
            while (v <= q && ckl >= b[v])
            {
                v++;
            }
            ksi = v - 1;
            points[k][l] = 0;
            for (i = 0; i <= p; i++)
            {
                points[k][l] = points[k][l] + (coeff[i][ksi][0] + coeff[i][ksi]
            }
            points[k][l] = rho * points[k][l];
            if (option == 1)
            {
                points[k][l] = MAX(points[k][l], ckl - K);
            }
            if (option == 2)
            {
                points[k][l] = MAX(points[k][l], K - ckl);
            }
        }
    }
}

/*computation of the coefficients*/

```

```

        solve(p, q, a, b, coeff, points);

    }

if (n != 2)
{
    /*computation of the approximated w1
    computation of the value at the points*/
    for (k = 0; k <= p; k++)
    {
        l = 0;
        ckl = a[k];
        v = 0;
        ksi = 0;
        while (v <= q && ckl >= b[v])
        {
            v++;
        }
        ksi = v - 1;
        points[k][l] = 0;
        for (i = 0; i <= p; i++)
        {
            points[k][l] = points[k][l] + (coeff[i][ksi][0] + coeff[i][ksi][2]
        }
        points[k][l] = rho * points[k][l];
        if (option == 1)
        {
            points[k][l] = MAX(points[k][l], ckl - K);
        }
        if (option == 2)
        {
            points[k][l] = MAX(points[k][l], K - ckl);
        }
    }
    /*computation of the coefficients*/
    for (k = 0; k <= p - 1; k++)
    {
        p1 = points[k][0];
        p2 = points[k + 1][0];
        coeff[k][0][1] = (p2 - p1) / (a[k + 1] - a[k]);
        coeff[k][0][0] = p1 - coeff[k][0][1] * a[k];
    }
}

```

```

    }
    l = 0;
    coeff[p][1][0] = coeff[p - 1][1][0];
    coeff[p][1][1] = coeff[p - 1][1][1];
    coeff[p][1][2] = coeff[p - 1][1][2];
    coeff[p][1][3] = coeff[p - 1][1][3];
}

/*computation of the approximated wo*/
k = 0;
while (So >= a[k] && k <= p)
{
    k++;
}
k = k - 1;
p1 = 0;
p2 = 0;
ck1 = 0;
points[k][0] = 0;
points[k + 1][0] = 0;
/*computation of the value at the points*/
for (i = 0; i <= p; i++)
{
    points[k][0] = points[k][0] + (coeff[i][0][0] + coeff[i][0][2] * ck1) * P[
}
points[k][0] = rho * points[k][0];
ck1 = 0;
for (i = 0; i <= p; i++)
{
    points[k + 1][0] = points[k + 1][0] + (coeff[i][0][0] + coeff[i][0][2] * c
}
points[k + 1][0] = rho * points[k + 1][0];
p1 = points[k][0];
p2 = points[k + 1][0];
/*computation of the coefficients*/
coeff[k][0][1] = (p2 - p1) / (a[k + 1] - a[k]);
coeff[k][0][0] = p1 - coeff[k][0][1] * a[k];
k = 0;
while (So >= a[k] && k <= p)
{
    k++;
}

```

```

    }
    k = k - 1;

    /*the result*/
    cc2 = coeff[k][0][1];
    cc0 = coeff[k][0][0];

    super = (n == 1) ? vnMoinsUn(So, 0, sigma, r, d, K, h, n, option) : (cc0 + So
    if (option == 1 || option == 3)
    {
        delta = (n == 1) ? exp(-d * h) * cdf_nor(d1) : cc2;
    }
    else
    {
        delta = (n == 1) ? -exp(-d * h) * cdf_nor(-d1) : cc2;
    }

    /*Price*/
    *ptprice = super;

    /*Delta */
    *ptdelta = delta;

    free(a);
    free(b);
    free(t);

    for (i = 0; i < p + 1; i++)
        free(P[i]);
    free(P);

    for (i = 0; i < p + 1; i++)
        free(Q[i]);
    free(Q);

    for (i = 0; i < p + 1; i++)
        free(points[i]);
    free(points);

    for (i = 0; i <= p; i++)

```

```

        for (j = 0; j <= q; j++)
            free(coeff[i][j]);
    for (i = 0; i <= p; i++)
        free(coeff[i]);
    free(coeff);

    return OK;
}

int CALC(FD_FixedAsian_BenHameurBretonLecuyer)(void *Opt, void *Mod, PricingMeth
{

    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    int return_value, am_or_eu;
    double r, divid, time_spent, pseudo_spot, pseudo_strike;
    double t_0, T_0;

    if ((ptOpt->EuOrAm).Val.V_BOOL == EURO)
        am_or_eu = 0;
    else am_or_eu = 1;

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

    T_0 = ptMod->T.Val.V_DATE;
    t_0 = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;

    if (T_0 < t_0)
    {
        Fprintf(TOSCREEN, "T_0 < t_0, untreated case\ n\ n\ n");
        return_value = WRONG;
    }
    /* Case t_0 <= T_0 */
    else

```

```

    {
        time_spent = (ptMod->T.Val.V_DATE - (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[
                    (ptOpt->Maturity.Val.V_DATE - (ptOpt->PathDep.Val.V_NUMFUNC_2
pseudo_spot = (1. - time_spent) * ptMod->S0.Val.V_PDOUBLE;
pseudo_strike = (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE - ti

        if (pseudo_strike <= 0.)
        {
            Fprintf(TOSCREEN, "ANALYTIC FORMULA\ n\ n\ n");
            return_value = Analytic_KemnaVorst(pseudo_spot, pseudo_strike, time_sp
        }
        else
            return_value = Fixed_BenHameurBretonLecuyer(am_or_eu, ptOpt->PayOff.Val.
    }
    return return_value;
}

static int CHK_OPT(FD_FixedAsian_BenHameurBretonLecuyer)(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 = 10;
        Met->Par[1].Val.V_INT2 = 50;
        Met->Par[2].Val.V_INT2 = 50;

    }

    return OK;
}

PricingMethod MET(FD_FixedAsian_BenHameurBretonLecuyer) =
{
    "FD_FixedAsian_BenHameurBretonLecuyer",

```

```

{"TimeStepNumber", INT2, {100}, ALLOW}, {"SpaceStepNumber1", INT2, {100}, ALL
CALC(FD_FixedAsian_BenHameurBretonLecuyer),
{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
CHK_OPT(FD_FixedAsian_BenHameurBretonLecuyer),
CHK_ok,
MET(Init)
};

```