

[Help](#)

```
#include "
href../../../../mod/bs1d/bs1d_pad/bs1d_pad_h_src.pdfbs1d_pad.h"
#include "
href../../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2009+2) //The "#els
static int CHK_OPT(AP_FixedAsian_LordUp)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_FixedAsian_LordUp)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else
static double m2_lord, m3_lord;

static void GaussLegendre_lord(double x1, double x2, double *x, double *w, int n
{

    int m;
    int j;
    int i;
    double z1, z, xm, x1, pp, p3, p2, p1;

    m = (np + 1) / 2;
    xm = 0.5 * (x2 + x1);
    x1 = 0.5 * (x2 - x1);

    for (i = 1; i <= m; i++)
    {
        z = cos(M_PI * (i - 0.25) / (np + 0.5));

        do
        {
            p1 = 1.0;
            p2 = 0.0;
            for (j = 1; j <= np; j++)
            {
```

```

        p3 = p2;
        p2 = p1;
        p1 = ((2.0 * j - 1.0) * z * p2 - (j - 1.0) * p3) / j;
    }
    pp = np * (z * p1 - p2) / (z * z - 1.0);
    z1 = z;
    z = z1 - p1 / pp;
}
while (fabs(z - z1) > 0.00000001);

x[i]      = xm - x1 * z;
x[np + 1 - i] = xm + x1 * z;

w[i]      = 2.0 * x1 / ((1.0 - z * z) * pp * pp);
w[np + 1 - i] = w[i];
}

}

//calculer l'integrale double d'une fonction a 4 variables
double integrale_double4_lord(double a, double b, int n1, double c, double d, int n2)
{
    double s = 0.;
    double *x, *w, *t, *y;
    int i;
    int j;

    x = malloc((n1 + 1) * sizeof(double));
    w = malloc((n1 + 1) * sizeof(double));
    t = malloc((n2 + 1) * sizeof(double));
    y = malloc((n2 + 1) * sizeof(double));

    GaussLegendre_lord(a, b, x, w, n1);
    GaussLegendre_lord(c, d, t, y, n2);

    for (i = 1; i < (n1) + 1; i++)
    {
        for (j = 1; j < (n2) + 1; j++)
        {
            s = s + w[i] * y[j] * fct(x[i], t[j], sigma, gamma1, S0, K, T, R, DIVI);
        }
    }
}

```

```

        }
    }
    free(x);
    free(w);
    free(t);
    free(y);

    return s;
}

//calculer l'integrale d'une fonction a 3 variables
double integrale3_lord(double a, double b, int n1, double y, double sigma, double K, double T, double R, double DIVID, double SIGMA)
{
    double s = 0.;

    int i;
    double *x, *w;

    x = malloc((n1 + 1) * sizeof(double));
    w = malloc((n1 + 1) * sizeof(double));
    GaussLegendre_lord(a, b, x, w, n1);

    for (i = 1; i < (n1) + 1; i++)
    {
        s = s + w[i] * fct(x[i], y, sigma, S0, K, T, R, DIVID, SIGMA);
    }

    free(x);
    free(w);
    return s;
}

// fonction qui trouve deux reels gauche et droite tel que nu(gauche)*nu(droite)
double bornage_nu_lord(double gauche, double droite, double S0, double K, double T, double R, double DIVID, double SIGMA)
{
    while (fct(gauche, S0, K, T, R, DIVID, SIGMA)*fct(droite, S0, K, T, R, DIVID, SIGMA) > 1)
    {
        gauche = gauche + 1;
    }
}

```

```

    return gauche;
}
// meme principe que bornage nu mais avec deux variables

double bornage2_lord(double gauche, double droite, double sigma, double S0, double K, double T, double R, double DIVID, double SIGMA)
{
    while (fct(gauche, sigma, S0, K, T, R, DIVID, SIGMA)*fct(droite, sigma, S0, K, T, R, DIVID, SIGMA) > 0)
    {
        gauche = gauche + 1;
    }
    return gauche;
}

//dichotomie trouve le zero d'une fonction a une variable
double dichotomie_lord(double a, double b, double S0, double K, double T, double R, double DIVID, double SIGMA)
{
    double gauche, droite, fg, fc, c;
    double precision = 0.00000001;

    int i;
    /* Initialisations */
    i = 0;
    gauche = a;
    droite = b;
    fg = fct(gauche, S0, K, T, R, DIVID, SIGMA) ;

    /* Boucle d'iteration */
    while ((droite - gauche) > precision)
    {
        c = (gauche + droite) / 2;

        i = i + 1;

        fc = fct(c, S0, K, T, R, DIVID, SIGMA);
        if (fg * fc < 0)
            droite = c;
        else
        {
            gauche = c;
            fg = fc;
        }
    }
}

```

```

    }
}

return (gauche + droite) / 2.;
}

//trouve le zero d'une fonction a deux variables avec l'un des deux parametres f
double dichotomie2_lord(double a, double b, double sigma, double S0, double K, d
{
    double gauche, droite, fg, fc, c;
    double precision = 0.00000001;

    int i;
    /* Initialisations */
    i = 0;
    gauche = a;
    droite = b;
    fg = fct(gauche, sigma, S0, K, T, R, DIVID, SIGMA) ;

    /* Boucle d'iteration */
    while ((droite - gauche) > precision)
    {
        c = (gauche + droite) / 2;

        i = i + 1;

        fc = fct(c, sigma, S0, K, T, R, DIVID, SIGMA);
        if (fg * fc < 0)
            droite = c;
        else
        {
            gauche = c;
            fg = fc;
        }

    }
    return (gauche + droite) / 2.;
}

```

```

// calcul des comatrices
void comatrices_lord(double a[3][3], double c[3][3], int i, int j, int n1)
{
    int l, k;
    for (l = 0; l < n1; l++) for (k = 0; k < n1; k++)
    {
        if ((l < i) && (k < j)) c[l][k] = a[l][k];
        if ((l > i) && (k < j)) c[l - 1][k] = a[l][k];
        if ((l < i) && (k > j)) c[l][k - 1] = a[l][k];
        if ((l > i) && (k > j)) c[l - 1][k - 1] = a[l][k];
    }
}

// calcul du determinant
double det_lord(double a[3][3], int n1)
{
    int k, j;
    double c[3][3], s;

    k = n1 - 1;

    if (n1 == 0) return (1);

    s = 0;
    for (j = 0; j < n1; j++)
    {
        comatrices_lord(a, c, k, j, n1);
        s = s + PNL_ALTERNATE(k + j) * a[k][j] * det_lord(c, k);
    }
    return (s);
}

//resolution par methode de cramer
void cramer_lord(double a[3][3], double b[3], double x[3], int n1)
{
    double A[3][3], deter;
    int i, j, k;

    deter = det_lord(a, n1);

```

```

if (deter == 0)
{
    printf("\n => Determinant nul, pas de solutions \n\n");
    system("PAUSE");
}

for (j = 0; j < n1; j++)
{
    for (k = 0; k < n1; k++)
    {
        if (k == j) for (i = 0; i < n1; i++) A[i][k] = b[i];
        else for (i = 0; i < n1; i++) A[i][k] = a[i][k];
    }
    x[j] = det_lord(A, n1) / deter;
}

/*trouver nu1 solution de nu(z)=0*/
static double nu_lord(double z, double S0, double K, double T, double R, double
{
    double y1 = (0.5 + sqrt(0.25 + m2_lord * exp(-2 * z)));
    return (exp(3 * z) * (pow(y1, 4.5) - 3 * pow(y1, 2.5) + 2 * pow(y1, 1.5)) - m3
}

//fonction qui calcule un Nu,w,alpha pour un t donne pour pouvoir calculer le ma
double cond_init_lord(double t, double y, double sigma, double S0, double K, dou
{
    double A1, A2, A3, m1, b3, nu1, w, alpha;
    A1 = pow(S0, 3) * exp(3 * (R - DIVID) * t) * (exp(3 * pow(SIGMA, 2) * t) - 3 *
    A2 = pow(S0 * t, 2) * (SIGMA / T) * exp(2 * (R - DIVID) * t) * (1 - exp(pow(SI
    A3 = pow(pow(t, 2) * SIGMA / T, 2) * S0 * exp((R - DIVID) * t) / 4;

    /*termes particulier*/
    m1 = S0 * exp((R - DIVID) * t);
    m2_lord = (pow(S0, 2) * exp(2 * (R - DIVID) * t) * (exp(pow(SIGMA, 2) * t) - 1
    m3_lord = A1 + 3 * K * sigma * A2 + 3 * pow(K * sigma, 2) * A3;

    b3 = bornage_nu_lord(-10, -10, S0, K, T, R, DIVID, SIGMA, nu_lord);

    nu1 = dichotomie_lord(-10, b3, S0, K, T, R, DIVID, SIGMA, nu_lord);

```

```

w = sqrt(log(0.5 + sqrt(0.25 + m2_lord * exp(-2 * nu1))));

alpha = m1 - exp(nu1 + pow(w, 2) / 2);
return (alpha + exp(nu1 + y * w));

}

//fonction issu des calculs de l'article
double f3_lord(double y, double sigma, double S0, double K, double T, double R,
{
    return
        integrale3_lord(0, T, 5 * T, y, sigma, S0, K, T, R, DIVID, SIGMA, cond_init_

}

//calcul de gamma pour trouver le mu1
double Gamma1_lord(double sigma, double S0, double K, double T, double R, double
{
    double b = bornage2_lord(-10, -10, sigma, S0, K, T, R, DIVID, SIGMA, f3_lord);
    return dichotomie2_lord(-10, b, sigma, S0, K, T, R, DIVID, SIGMA, f3_lord);
}

//l'esperance optimisee
double mu1_lord(double t, double sigma, double gamma1, double S0, double K, double
{
    return (1 / K) * cond_init_lord(t, gamma1, sigma, S0, K, T, R, DIVID, SIGMA);
}

double g5(double y, double x, double sigma, double gamma1, double S0, double K,

{
    double A = (R - DIVID - pow(SIGMA, 2) / 2);
    double at = (S0 * exp(SIGMA * y * x + A * pow(y, 2)) - K * mu1_lord(pow(y, 2),
    double bt = (K * sigma) * sqrt((T / 3) - pow(y, 2) * pow(1 - pow(y, 2) / (2 *

    return (2 * y * pnl_normal_density(x) * (at * cdf_nor(at / bt) + bt * pnl_norm
}

```



```

//majorant du prix
double UB1_lord(double sigma, double gamma1, double S0, double K, double T, double R, double DIVID, double SIGMA)
{
    return (exp(-R * T) / T) * integrale_double4_lord(0, sqrt(T), 5 * T, -6, 6, 60, sigma, gamma1, S0, K, T, R, DIVID, SIGMA);
}

//fonction qui remplit une matrice et un vecteur par des valeurs donnees pour de
void init1_lord(double a, double b, double g, double m[3][3], double c[3], double S0, double K, double T, double R, double DIVID, double SIGMA)
{
    int i;

    for (i = 0; i < 3; i++)
    {
        m[0][i] = pow(a, (2 - i));
    }
    for (i = 0; i < 3; i++)
    {
        m[1][i] = pow(b, (2 - i));
    }
    for (i = 0; i < 3; i++)
    {
        m[2][i] = pow(g, (2 - i));
    }
    c[0] = UB1_lord(a, Gamma1_lord(a, S0, K, T, R, DIVID, SIGMA), S0, K, T, R, DIVID, SIGMA);
    c[1] = UB1_lord(b, Gamma1_lord(b, S0, K, T, R, DIVID, SIGMA), S0, K, T, R, DIVID, SIGMA);
    c[2] = UB1_lord(g, Gamma1_lord(g, S0, K, T, R, DIVID, SIGMA), S0, K, T, R, DIVID, SIGMA);
}

//deuxieme majorant en trouvant une parabole passant par trois points et en calculant
static double SLNQuad1_lord(double a, double b, double g, double S0, double K, double T, double R, double DIVID, double SIGMA)
{
    double x[3];
    double m[3][3];
    double c[3];
    double min;

    init1_lord(a, b, g, m, c, S0, K, T, R, DIVID, SIGMA);
    cramer_lord(m, c, x, 3);

    min = UB1_lord(-x[1] / (2 * x[0]), Gamma1_lord(-x[1] / (2 * x[0]), S0, K, T, R, DIVID, SIGMA));
}

```

```

    return min;
}

static int LordUp_FixedAsian(double S0, double K, NumFunc_2 *po, double T, double R, double DIVID, double SIGMA)
{
    double inc;
    double CTtK, CTtK_inc, PTtK, Dlt, Plt;

    /*Increment for the Delta*/
    inc = 1.0e-3;

    if (flag == 1)
    {
        double gamma_SLN = Gamma1_lord(SIGMA, S0, K, T, R, DIVID, SIGMA);

        /*Call Price */
        CTtK = UB1_lord(SIGMA, gamma_SLN, S0, K, T, R, DIVID, SIGMA);
        CTtK_inc = UB1_lord(SIGMA, gamma_SLN, S0 * (1. + inc), K, T, R, DIVID, SIGMA);
    }
    else
    {
        /* Call Price */
        CTtK = SLNQuad1_lord(0.5 * SIGMA, 0.75 * SIGMA, SIGMA, S0, K, T, R, DIVID, SIGMA);
        CTtK_inc = SLNQuad1_lord(0.5 * SIGMA, 0.75 * SIGMA, SIGMA, S0 * (1. + inc), K, T, R, DIVID, SIGMA);
    }

    /* Put Price from Parity */
    if (R == DIVID)
        PTtK = CTtK + K * exp(-R * T) - S0 * exp(-R * T);
    else
        PTtK = CTtK + K * exp(-R * T) - S0 * exp(-R * T) * (exp((R - DIVID) * T) - 1);

    /*Delta for call option*/
    Dlt = (CTtK_inc - CTtK) / (S0 * inc);

    /*Delta for put option */
    if (R == DIVID)
        Plt = Dlt - exp(-R * T);
    else

```

```

    Plt = Dlt - exp(-R * T) * (exp((R - DIVID) * T) - 1.0) / (T * (R - DIVID));

/*Price*/
if ((po->Compute) == &Call_OverSpot2)
    *ptprice = CTtK;
else
    *ptprice = PTtK;

/*Delta */
if ((po->Compute) == &Call_OverSpot2)
    *ptdelta = Dlt;
else
    *ptdelta = Plt;

return OK;
}

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

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

    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[
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 = LordUp_FixedAsian(pseudo_spot, pseudo_strike, ptOpt->PayO
}

return return_value;
}

static int CHK_OPT(AP_FixedAsian_LordUp)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedEuro") == 0) || (strcmp(((Op
        return OK;
    return WRONG;
}
#endif //PremiaCurrentVersion

static PremiaEnumMember ComputationMethodUpMembers[] =
{
    {"Upper Bound", 1},
    { "Shifted Log Normal Quad", 2},
    { NULL, NULLINT }
};

static DEFINE_ENUM(ComputationMethodUp, ComputationMethodUpMembers);

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->Par[0].Val.V_ENUM.value = 1;
        Met->Par[0].Val.V_ENUM.members = &ComputationMethodUp;
    }
}

```

```

    }

    return OK;
}

PricingMethod MET(AP_FixedAsian_LordUp) =
{
    "AP_FixedAsian_LordUp",
    { {"Conditioning Method", ENUM, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID},
    CALC(AP_FixedAsian_LordUp),
    {"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PREMIA_NULLTYPE, {0}, FORBID},
    CHK_OPT(AP_FixedAsian_LordUp),
    CHK_ok,
    MET(Init)
};

```