

[Help](#)

```

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

#define NUMNODI 100

static dcomplex ltftasia(dcomplex mu, dcomplex v, dcomplex n)
{
    dcomplex    num, nterm1, nterm2, nterm3;
    dcomplex    den, dterm1, dterm2;

    nterm1 = Clgamma(Cadd(n, CUNO));
    nterm2 = Clgamma(Cadd(RCmul(0.5, Cadd(mu, v)), CUNO));
    nterm3 = Clgamma(Csub(RCmul(0.5, Csub(mu, v)), n));
    num = Cadd(Cadd(nterm1, nterm2), nterm3);

    dterm1 = Clgamma(RCmul(0.5, Csub(mu, v)));
    dterm2 = Clgamma(Cadd(Cadd(RCmul(0.5, Cadd(mu, v)), CUNO), n));

    den = Cadd(RCmul(log(2.0), n), Cadd(dterm1, dterm2));

    return Cexp(Csub(num, den));
}

static dcomplex transformasia(dcomplex l, dcomplex g, double r, double sg)
{
    dcomplex    v, v2, mu, n, cimm, den;

    cimm = Complex(0.0, 1.0);
    n = Cadd(CUNO, Cmul(cimm, g));

    v = Complex(2 * r / (sg * sg) - 1.0, 0.0);
    v2 = Complex((2 * r / (sg * sg) - 1) * (2 * r / (sg * sg) - 1), 0.0);
    mu = Csqrt(Cadd(v2, RCmul(2.0, 1)));

    den = Cmul(cimm, Cmul(1, g));

```

```

    den = RCmul(sg * sg, Cmul(den, Cadd(CUNO, Cmul(cimm, g))));
    den = Cmul(1, RCmul(sg * sg, Cmul(Csub(n, CUNO), n)));

    return Cdiv(RCmul(4.0, ltftasia(mu, v, n)), den);
}

dcomplex infasia(dcomplex l, double logstrike, double r, double sg,
                double aaf, int termsf, int tottermsf)
{
    int j;
    double pg = 3.14159265358979358;

    dcomplex term1, term2, term, Eulero;
    dcomplex sum;
    double *sum_r, *sum_i;

    /*Memory Allocation*/
    sum_r = malloc((tottermsf - termsf + 1 + 1) * sizeof(double));
    sum_i = malloc((tottermsf - termsf + 1 + 1) * sizeof(double));

    sum = Complex(0.0, 0.0);
    Eulero = Complex(0.0, 0.0);

    sum = transformasia(l, Complex(0, aaf / (2 * logstrike)), r, sg);

    for (j = 1; j <= tottermsf; j++)
    {
        term1 = RCmul(POW(-1.0, j) , transformasia(l, Complex(j * pg / logstrike,
        term2 = RCmul(POW(-1.0, j) , transformasia(l, Complex(-j * pg / logstrike,
        term = Cadd(term1, term2);

        sum = Cadd(term, sum);

        if (termsf <= j) sum_r[j - termsf + 1] = sum.r;
        if (termsf <= j) sum_i[j - termsf + 1] = sum.i;
    }

    for (j = 0; j <= tottermsf - termsf; j++)
    {
        Eulero.r = Eulero.r + bico(tottermsf - termsf, j) * POW(2.0, -(tottermsf -
        Eulero.i = Eulero.i + bico(tottermsf - termsf, j) * POW(2.0, -(tottermsf -

```

```

    }

    free(sum_r);
    free(sum_i);

    return RCmul(exp(aaf / 2) / (2 * logstrike), Euler);
}

static dcomplex transformDeltaasia(dcomplex l, dcomplex g, double r, double sg,
{
    dcomplex cimm, ig ;

    cimm = Complex(0.0, 1.0);

    ig = Cadd(CUN0, Cmul(cimm, g));
    return Cmul(ig, transformasia(l, g, r, sg));
}

static dcomplex infDeltaasia(dcomplex l, double logstrike, double r, double sg,
                             double aaf, int termsf, int tottermsf)
{
    int j;
    double pg = 3.14159265358979358;

    dcomplex term1, term2, term, Euler;
    dcomplex sum;
    double *sum_r, *sum_i;

    /*Memory Allocation*/
    sum_r = malloc((tottermsf - termsf + 1 + 1) * sizeof(double));
    sum_i = malloc((tottermsf - termsf + 1 + 1) * sizeof(double));

    sum = Complex(0.0, 0.0);
    Euler = Complex(0.0, 0.0);

    sum = transformDeltaasia(l, Complex(0, aaf / (2 * logstrike)), r, sg);

```

```

for (j = 1; j <= tottermsf; j++)
{
    term1 = RCmul(POW(-1.0, j) , transformDeltaasia(1, Complex(j * pg / logstr
    term2 = RCmul(POW(-1.0, j) , transformDeltaasia(1, Complex(-j * pg / logst
    term = Cadd(term1, term2);

    sum = Cadd(term, sum);

    if (termsf <= j)  sum_r[j - termsf + 1] =  sum.r;
    if (termsf <= j)  sum_i[j - termsf + 1] =  sum.i;
}

for (j = 0; j <= tottermsf - termsf; j++)
{
    Eulero.r = Eulero.r + bico(tottermsf - termsf, j) * POW(2.0, -(tottermsf -
    Eulero.i = Eulero.i + bico(tottermsf - termsf, j) * POW(2.0, -(tottermsf -
}

free(sum_r);
free(sum_i);

return RCmul(exp(aaf / 2) / (2 * logstrike), Eulero);
}

static int LaplaceFourier_FixedAsian(double spot, double strike, NumFunc_2 *po,
{

    int k;
    double pg = 3.14159265358979358;
    double h, Eulero, logstrike;
    dcomplex term, sum;
    double *sum_r;
    double invlaplace;
    double aaf = 22.4;
    int tottermsf;
    double aa = 22.4;
    int totterms;
    double CTtK, PTtK, Dlt, Plt;
    double alpha = 0.;

```

```

tottermsf = termsf + 15;
totterms = terms + 15;
h = sg * sg * expiry / 4.0;
logstrike = log(strike * h / spot);
sum_r = malloc((tottermsf - termsf + 1 + 1) * sizeof(double));

sum = Complex(0.0, 0.0);
Eulero = 0.0;

sum = RCmul(1.0 / 2.0, infasia(Complex(aa / (2.0 * h), 0), logstrike, r, sg,
for (k = 1; k <= totterms; k++)
{
    term = RCmul(POW(-1.0, k) , infasia(Complex(aa / (2.0 * h) , k * pg / h),

    sum = Cadd(term, sum);

    if (terms <= k) sum_r[k - terms + 1] = sum.r;
}

for (k = 0; k <= totterms - terms; k++)
{
    Eulero = Eulero + bico(totterms - terms, k) * POW(2.0, -(totterms - terms)

}

free(sum_r);

invlaplace = exp(aa / 2.0) * Eulero / h;

/* Call Price */
CTtK = -exp(-r * expiry - alpha * logstrike) * spot * invlaplace / (expiry);

/* Put Price from Parity*/
if (r == divid)
    PTtK = CTtK + strike * exp(-r * expiry) - spot * exp(-r * expiry);
else
    PTtK = CTtK + strike * exp(-r * expiry) - spot * exp(-r * expiry) * (exp((r

/*Delta Computation*/

```

```

sum_r = malloc((tottermsf - termsf + 1 + 1) * sizeof(double));
sum = Complex(0.0, 0.0);
Eulero = 0.0;

sum = RCmul(1.0 / 2.0, infDeltaasia(Complex(aa / (2.0 * h), 0), logstrike, r,

for (k = 1; k <= totterms; k++)
{
    term = RCmul(POW(-1.0, k) , infDeltaasia(Complex(aa / (2.0 * h) , k * pg /

    sum = Cadd(term, sum);

    if (terms <= k) sum_r[k - terms + 1] = sum.r;
}

for (k = 0; k <= totterms - terms; k++)
{
    Eulero = Eulero + bico(totterms - terms, k) * POW(2.0, -(totterms - terms)
}
free(sum_r);
invlaplace = exp(aa / 2.0) * Eulero / h;

/*Delta for call option*/
Dlt = -exp(-r * expiry - alpha * logstrike) * invlaplace / (expiry);

/*Delta for put option*/
if (r == divid)
    Plt = Dlt - exp(-r * expiry);
else
    Plt = Dlt - exp(-r * expiry) * (exp((r - divid) * expiry) - 1) / (expiry * (

/*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_LaplaceFourier)(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 ((ptMod->Divid.Val.V_DOUBLE > 0))
    {
        Fprintf(TOSCREEN, "Divid >0 , untreated case\ n\ n\ n");
        return_value = WRONG;
    }
    else 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[0].Val.V_DATE) / 360.;
        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 = LaplaceFourier_FixedAsian(pseudo_spot, pseudo_strike, pt0
    }

    return return_value;
}

static int CHK_OPT(AP_FixedAsian_LaplaceFourier)(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 = 315;
        Met->Par[1].Val.V_INT2 = 315;

    }

    return OK;
}

PricingMethod MET(AP_FixedAsian_LaplaceFourier) =
{
    "AP_FixedAsian_LaplaceFourier",
    { {"Euler Fourier Terms", INT2, {2000}, ALLOW },
      {"Euler Laplace Terms", INT2, {1000}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(AP_FixedAsian_LaplaceFourier),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID} ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}

```



```
    },  
    CHK_OPT(AP_FixedAsian_LaplaceFourier),  
    CHK_ok,  
    MET(Init)  
};
```