

[Help](#)

```
#include "
href../../../../mod/ouid/ouid_std/ouid_std_h_src.pdfouid_std.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_root.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2013+2) //The "#els
static int CHK_OPT(AP_FOURIERCOSINE_OU)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_FOURIERCOSINE_OU)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static PnlVectComplex *be;
static int Nnumber, gflag, gM;
static double ga, gb, gT, gr, gstrike;

void static funcg(double t, double a1, double a2, double a3, double *g)
{
    *g = log(a1 + a2 * sin(a3 * t));
}
void static funca(double u, double t1, double t2, double kappa, double sigma, do
{
    double g1, g2;
    funcg(t1, a1, a2, a3, &g1);
    funcg(t2, a1, a2, a3, &g2);
    *aa = RCadd(u * u * sigma * sigma / (4.*kappa) * (exp(-2.*kappa * (t2 - t1)) -
}
void static meanxt(double t, double S0, double kappa, double sigma, double a1, d
{
```

```

double gt, g0;
funcg(t, a1, a2, a3, &gt);
funcg(0, a1, a2, a3, &g0);
*mean = log(S0) * exp(-kappa * t) + gt - g0 * exp(-kappa * t);
}
void static variancext(double t1, double t2, double kappa, double sigma, double
{
    *variance = sigma * sigma / (2 * kappa) * (1. - exp(-2.*kappa * (t2 - t1)));
}
void static rangex(double t, double S0, double kappa, double sigma, double a1, d
{
    double mean, variance, L;
    meanxt(t, S0, kappa, sigma, a1, a2, a3, &mean);
    variancext(0, t, kappa, sigma, &variance);
    L = 12.;
    *a = mean - L * variance;
    *b = mean + L * variance;
}
void static fxi(double a, double b, double l, double u, int n, double *result)
{
    *result = (n == 0) ? (exp(u) - exp(l)) : 1. / (1. + pow(n * M_PI / (b - a) , 2
}
void static fpsi(double a, double b, double l, double u, int n, double *result)
{
    *result = (n == 0) ? (u - l) : (sin(n * M_PI * (u - a) / (b - a)) - sin(n * M_
}
void static fV(double a, double b, double strike, int n, int m, int maturity, in
{
    double xi, psi, l, u, result = 0.;
    if (flag == -1) //put option
    {
        if (m == maturity)
        {
            fxi(a, b, a, 0., n, &xi);
            fpsi(a, b, a, 0., n, &psi);
            result = 2. / (b - a) * flag * strike * (xi - psi);
        }
        else
        {
            u = (GET(star, m + 1) <= 0.) ? GET(star, m + 1) : 0.;
            l = (a <= 0.) ? a : 0.;

```

```

        fxi(a, b, l, u, n, &xi);
        fpsi(a, b, l, u, n, &psi);
        result = GET(chat, n) + 2. / (b - a) * flag * strike * (xi - psi);
    }
}
else if (flag == 1) // call option
{
    if (m == maturity)
    {
        fxi(a, b, 0., b, n, &xi);
        fpsi(a, b, 0., b, n, &psi);
        result = 2. / (b - a) * flag * strike * (xi - psi);
    }
    else
    {
        u = (b >= 0.) ? b : 0.;
        l = (GET(star, m + 1) >= 0.) ? GET(star, m + 1) : 0.;
        fxi(a, b, l, u, n, &xi);
        fpsi(a, b, l, u, n, &psi);
        result = GET(chat, n) + 2. / (b - a) * flag * strike * (xi - psi);
    }
}
pnl_vect_set(ve, n, result);
}
void static fphi(double a, double b, double S0, double kappa, double sigma, double
{
    int k;
    dcomplex aa;
    double mean;
    for (k = 0; k < number; k++)
    {
        funca(k * M_PI / (b - a), m * deltat, (m + 1)*deltat, kappa, sigma, a1, a2
        meanxt(m * deltat, S0, kappa, sigma, a1, a2, a3, &mean);
        pnl_mat_complex_set(tphi, m, k, Cexp(Cadd(CRmul(CI, -k * M_PI / (b - a)*me
    }
}
void static funcbeta(int k, int m, PnlMatComplex *tphi, PnlVect *v)
{
    pnl_vect_complex_set(be, k, CRmul(pnl_mat_complex_get(tphi, m, k), pnl_vect_g
}
static void fdf(double x, double *f, double *df, void *p)

```

```

{
    int n;
    dcomplex sum = CZERO, sumd = CZERO;
    double g = 0.0, dg = 0.0;
    sum = RCmul(0.5, pnl_vect_complex_get(be, 0));
    sumd = CZERO;
    for (n = 1; n < Nnumber; n++)
    {
        sum = Cadd(sum, Cmul(pnl_vect_complex_get(be, n), Cexp(CRmul(CI, n * M_PI
        sumd = Cadd(sumd, Cmul(pnl_vect_complex_get(be, n), Cmul(Cexp(CRmul(CI, n
    }
    g = (gflag * (exp(x) - 1.0) >= 0.0) ? gflag * (exp(x) - 1.0) : 0.0;
    *f = exp(-gr * gT / gM) * Creal(sum) - gstrike * g ;
    dg = (gflag * (exp(x) - 1.0) >= 0.0) ? gflag * exp(x) : 0.0;
    *df = exp(-gr * gT / gM) * Creal(sumd) - gstrike * dg;
    if (fabs(exp(-gr * gT / gM)*Creal(sumd) - gstrike * dg) < 1E-9)
    {
        fprintf(stderr, "derivative is zero!!    sumd=%f+%f, dg=%f, %f-%f=%f, df=%f
    }
}

void static newton_root_find(int m, PnlVect *star)
{
    double x0, r;
    PnlFuncDFunc func;
    static double epsabs = 0.0000001;
    static double epsrel = 0.00000001;
    static int N_max = 10;

    if (m == gM)        x0 = 0.0;
    else    x0 = GET(star, m + 1);
    func.F = fdf;
    func.params = NULL;

    pnl_root_newton(&func, x0, epsrel, epsabs, N_max, &r);
    LET(star, m) = r;
}

void static mbasic(double a, double b, double l, double u, PnlVectComplex *mj)
{
    int n;
    pnl_vect_complex_set(mj, 0, RCmul((u - l)*M_PI / (b - a), CI));
    for (n = 1; n < 2 * Nnumber + 1; n++)

```

```

    {
        pnl_vect_complex_set(mj, n, CRdiv(Csub(Cexp(RCmul(n * (u - a)*M_PI / (b -
    }
}
void static c_fft(int number, double T, double M, double a, double b, double l,
{
    PnlVectComplex *ms = pnl_vect_complex_create(2 * number);
    PnlVectComplex *mc = pnl_vect_complex_create(2 * number);
    PnlVectComplex *us = pnl_vect_complex_create(2 * number);
    PnlVectComplex *ivector1 = pnl_vect_complex_create(2 * number);
    PnlVectComplex *ivector2 = pnl_vect_complex_create(2 * number);
    int n;

    for (n = 0; n < number; n++)
    {
        pnl_vect_complex_set(ms, n, RCmul(-1, Conj(pnl_vect_complex_get(mj, n))));
        pnl_vect_complex_set(us, n, pnl_vect_complex_get(beta, n));
        pnl_vect_complex_set(mc, n, pnl_vect_complex_get(mj, 2 * number - 1 - n));
    }
    pnl_vect_complex_set(us, 0, RCmul(0.5, pnl_vect_complex_get(beta, 0)));
    for (n = Nnumber; n < 2 * number; n++)
    {
        pnl_vect_complex_set(ms, n, pnl_vect_complex_get(mj, 2 * number - n));
        pnl_vect_complex_set(us, n, CZERO);
        pnl_vect_complex_set(mc, n, pnl_vect_complex_get(mj, 2 * number - 1 - n));
    }
    pnl_vect_complex_set(ms, number, CZERO);

    pnl_fft_inplace(us);
    pnl_fft_inplace(ms);
    pnl_fft_inplace(mc);

    for (n = 0; n < 2 * number; n++)
    {
        pnl_vect_complex_set(ivector1, n, Cmul(pnl_vect_complex_get(ms, n), pnl_ve
        pnl_vect_complex_set(ivector2, n, Cmul(pnl_vect_complex_get(mc, n), pnl_ve
    }
    for (n = 0; n < number; n++)
    {
        pnl_vect_complex_set(ivector2, 2 * n + 1, RCmul(-1., pnl_vect_complex_get(
    }

```

```

pnl_ifft_inplace(ivector1);
pnl_ifft_inplace(ivector2);
for (n = 0; n < number; n++)
{
    pnl_vect_set(chat, n, exp(-r * T / M) / M_PI * (Cimag(pnl_vect_complex_get

pnl_vect_complex_free(&ms);
pnl_vect_complex_free(&mc);
pnl_vect_complex_free(&us);
pnl_vect_complex_free(&ivector1);
pnl_vect_complex_free(&ivector2);
}

void static commodity_ou(double S0, double strike, double r, double T, double ka
{
    int m, n;
    double a, b;
    dcomplex sum = CZERO;
    PnlVect *v = pnl_vect_create_from_zero(number);
    PnlMatComplex *phi = pnl_mat_complex_create(M, number);
    PnlVect *chat = pnl_vect_create_from_zero(number);
    PnlVect *star = pnl_vect_create_from_zero(number + 1);
    PnlVectComplex *mj = pnl_vect_complex_create(2 * number + 1);

    be = pnl_vect_complex_create(number);
    rangex(T, S0, kappa, sigma, a1, a2, a3, &a, &b);
    a = a - log(strike);
    b = b - log(strike);
    ga = a;
    gb = b;
    for (n = 0; n < number; n++)
    {
        fV(a, b, strike, n, M, M, flag, chat, star, v);
    }
    for (m = 0; m < M; m++)
    {
        fphi(a, b, S0, kappa, sigma, a1, a2, a3, T / M, m, number, phi);
    }
    for (n = 0; n < number; n++)
    {
        funcbeta(n, M - 1, phi, v);
    }
}

```

```

    }
    newton_root_find(M, star);
    mbasic(a, b, GET(star, M), b, mj);
    c_fft(number, T, M, a, b, GET(star, M) , b, r, mj, be, chat);
    for (m = M - 1; m > 1; m--)
    {
        for (n = 0; n < number; n++)
        {
            fV(a, b, strike, n, m, M, flag, chat, star, v);
        }
        for (n = 0; n < number; n++)
        {
            funcbeta(n, m - 1, phi, v);
        }
        newton_root_find(m, star);
        mbasic(a, b, GET(star, m), b, mj);
        c_fft(number, T, M, a, b, GET(star, m) , b, r, mj, be, chat);
    }
    fV(a, b, strike, 0, 1, M, flag, chat, star, v);
    funcbeta(0, 0, phi, v);
    sum = RCmul(0.5, Cmul(pnl_vect_complex_get(be, 0), CONE));
    for (n = 1; n < number; n++)
    {
        fV(a, b, strike, n, 1, M, flag, chat, star, v);
        funcbeta(n, 0, phi, v);
        sum = Cadd(sum, Cmul(pnl_vect_complex_get(be, n), Cexp(CRmul(CI, n * M_PI
    }
    *price = exp(-r * T / M) * Creal(sum);

    //Free
    pnl_vect_free(&v);
    pnl_mat_complex_free(&phi);
    pnl_vect_free(&chat);
    pnl_vect_free(&star);
    pnl_vect_complex_free(&mj);
    pnl_vect_complex_free(&be);

}

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

```

```

static int ap_fouriercosine_ou(NumFunc_1 p, double S0, double T, double strike,
                               double r, double kappa, double sigma,
                               double a1, double a2, double a3, int flag, int M,
{
    //parameters for the method: N for COS series
    int N;
    double price;

    N = pow(2, 7.);

    gflag = flag;
    gstrike = strike;
    gT = T;
    gr = r;
    gM = M;
    Nnumber = N;

    commodity_ou(S0, strike, r, T, kappa, sigma, a1, a2, a3, M, flag, N, &price);

    *ptprice = price;

    return OK;
}

//=====
int CALC(AP_FOURIERCOSINE_OU)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, strike;
    NumFunc_1 *p;
    int res;
    int iscall;

    iscall = -1; //Put Case
    if (ptOpt->PayOff.Val.V_NUMFUNC_1->Compute == &Call) iscall = 1;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    p = ptOpt->PayOff.Val.V_NUMFUNC_1;
    strike = p->Par[0].Val.V_DOUBLE;

```



```

    res = ap_fouriercosine_ou(*p, ptMod->S0.Val.V_PDOUBLE, ptOpt->Maturity.Val.V_D
    return res;

}

static int CHK_OPT(AP_FOURIERCOSINE_OU)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PutAmer") == 0))
        return OK;

    return WRONG;
}
#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->Par[0].Val.V_INT = 100;
        Met->init = 1;
        Met->HelpFilenameHint = "ap_fouriercosine_ou";
    }
    return OK;
}

PricingMethod MET(AP_FOURIERCOSINE_OU) =
{
    "AP_FOURIERCOSINE_OU",
    {"Bermudan Steps", INT, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_FOURIERCOSINE_OU),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_FOURIERCOSINE_OU),
    CHK_split,
    MET(Init)
};

```

