

[Help](#)

```
#include "
href../../mod/kould/kould_pad/kould_pad_h_src.pdfkould_pad.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_vector.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(FFT_Kou_FixedLookback)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(FFT_Kou_FixedLookback)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

// "resultat takes the product of a circulant matrix M(c) and a vector x
static void circulante(PnlVectComplex *resultat, PnlVectComplex *c, PnlVectComplex *x)
{
    PnlVectComplex *temp;
    int i, n;
    n = x->size;
    temp = pnl_vect_complex_create(n);

    pnl_fft(c, temp);
    pnl_fft(x, resultat);
    for (i = 0; i < n; i++)
    {
        pnl_vect_complex_set(temp, i, Cmul(pnl_vect_complex_get(temp, i), pnl_vect_complex_get(resultat, i)));
    }
    pnl_ifft(temp, resultat);
    pnl_vect_complex_free(&temp);
}

// "r" takes the product of the toeplitz matrix N(v,w) with holes and a vector
```

```

static void toep(PnlVectComplex *v, PnlVectComplex *w, PnlVectComplex *x, PnlVectComplex *r)
{
    int M, i;
    PnlVectComplex *temp;
    PnlVectComplex *temp2;
    PnlVectComplex *x2;
    M = v->size;
    temp = pnl_vect_complex_create(4 * M);
    temp2 = pnl_vect_complex_create(4 * M);
    x2 = pnl_vect_complex_create(4 * M);
    pnl_vect_complex_set(temp, 0, CRmul(CONE, 0.5));
    for (i = 1; i < 2 * M + 1; i = i + 2)
    {
        pnl_vect_complex_set(temp, i, pnl_vect_complex_get(v, (i - 1) / 2));
        pnl_vect_complex_set(temp, i + 1, CZERO);
    }
    for (i = 2 * M + 1; i < 4 * M - 1; i = i + 2)
    {
        pnl_vect_complex_set(temp, i, pnl_vect_complex_get(w, (M - 1) - ((i - 1) / 2)));
        pnl_vect_complex_set(temp, i + 1, CZERO);
    }
    pnl_vect_complex_set(temp, i, pnl_vect_complex_get(w, (M - 1) - ((4 * M - 1 - i) / 2)));
    for (i = 0; i < 2 * M + 1; i++)
        pnl_vect_complex_set(x2, i, pnl_vect_complex_get(x, i));
    for (i = 2 * M + 1; i < 4 * M; i++)
        pnl_vect_complex_set(x2, i, CZERO);
    circulante(temp2, temp, x2);
    for (i = 0; i < 2 * M + 1; i++)
        pnl_vect_complex_set(r, i, pnl_vect_complex_get(temp2, i));

    pnl_vect_complex_free(& temp);
    pnl_vect_complex_free(& temp2);
    pnl_vect_complex_free(& x2);
}

```

```

// the characteristic function of the Kou's model
static dcomplex fi_kou(double t, double sigma, double d, double lambda, double l)
{
    dcomplex temp1, temp2, u;

```

```

u = CRMul(w, (double)signe);
temp1 = CRMul(CRsub(Cmul(CI, u), lambdam), -1); //temp1=(I*u-lambdam)*(-1)
temp1 = Conj(temp1); //temp1=temp1.conj();
temp2 = CRadd(Cmul(CI, u), lambdap); //temp2=I*u+lambdap;
temp2 = Conj(temp2); //temp2=temp2.conj();
temp1 = CRdiv(temp1, Csqr_norm(temp1)); //temp1=(temp1/temp1.norm());
temp2 = CRdiv(temp2, Csqr_norm(temp2)); //temp2=temp2/(temp2.norm());
temp1 = Cadd(CRmul(temp1, lambdam * p), CRsub(CRmul(temp2, lambdap * (1 - p)),
temp1 = CRMul(temp1, -1); //temp1=temp1*-1;
temp1 = CRMul(temp1, lambda); //temp1=temp1*1;
temp2 = Csub(CRdiv(CRmul(Cmul(u, u), sigma * sigma), 2.), CRMul(Cmul(CI, u), d
temp1 = Cadd(temp2, temp1); //temp1=temp2+temp1;

temp1 = CRMul(temp1, (-t)); //temp1=temp1*(-t);
temp1 = Cexp(temp1); //temp1=temp1.expon();

return temp1;
}

```

```

// the characteristic function of the Kou's model after the Esscher transformati
static dcomplex fi_kou_star(double t, double sigma, double d, double lambda, dou
{
dcomplex b;
dcomplex a;
dcomplex c;
a = CRMul(CI, -1);
c = fi_kou(t, sigma, d, lambda, lambdap, lambdam, p, Cadd(u, CRMul(a, x)), sig
b = fi_kou(t, sigma, d, lambda, lambdap, lambdam, p, CRMul(a, x) , signe);
a = Cdiv(c, b);

return a;
}

```

```

//The diagonal matrix which attributes the corresponding coefficients to the calc
static void F_diag-fi_kou(PnlVectComplex *v, double t, double sigma, double d,
{
int com;

for (com = 0; com < 2 * M + 1; com++)

```

```

    pnl_vect_complex_set(v, com, Cmul(pnl_vect_complex_get(v, com), fi_kou_star(
}

```

```

//The diagonal matrix which attributes the corresponding coefficients to the calc
static void G_diag_fi_kou(PnlVectComplex *v, double t, double sigma, double d, d
{

```

```

    int i;

    for (i = 0; i < 2 * M + 1; i++)
        pnl_vect_complex_set(v, i, Cmul(pnl_vect_complex_get(v, i), fi_kou_star(t, s
}

```

```

// the estimation's algorithm in the Kou's model with "n_point" maximum observ
static dcomplex estiation_kou(double Xmaxmin, PnlVectComplex *G, double t, doubl
{

```

```

    int i, j;
    double Xmax;
    dcomplex C, cte, a, k;
    PnlVectComplex *v, *w, *F, *tempg, *tempf;
    F = pnl_vect_complex_create(2 * M + 1);
    v = pnl_vect_complex_create(M); //the first column vector of the Hilbert's
    w = pnl_vect_complex_create(M); //the first line vector of the Hilbert's ma
    tempg = pnl_vect_complex_create(2 * M + 1);
    tempf = pnl_vect_complex_create(2 * M + 1);
    //initialisation of v and w
    cte = CONE;
    C = CRmul(CI, M_1_PI); // 1/pi
    for (i = 0; i < M; i = i + 1)
        pnl_vect_complex_set(v, i, CRdiv(C, (double)(2 * i + 1)));
    for (i = 0; i < M; i = i + 1)
        pnl_vect_complex_set(w, i, CRdiv(C, (double)(-(2 * i + 1))));
    //intialisation of G_1, F_1 and Cte_1
    Xmax = MAX(Xmaxmin, 0);
    for (i = 0; i < 2 * M + 1; i++)
    {
        pnl_vect_complex_set(F, i, Cexp(Cmul(CRmul(CRmul(h, (double)(i - M)) , Xma
        pnl_vect_complex_set(G, i, Cexp(CRmul(CRadd(Cmul(CRmul(h, (double)(i - M))
    }
    F_diag_fi_kou(F, t, sigma, d, lambda, lambdap, lambdam, p, h, x, M, signe); //
    toep(v, w, F, tempf);

```

```

pnl_vect_complex_clone(F, tempf);

cte = CRMul(CRsub(pnl_vect_complex_get(F, M), 1.0), -1); //1-f(0)

G_diag_fi_kou(G, t, sigma, d, lambda, lambdap, lambdam, p, h, x, M, signe); //
toep(v, w, G, tempg); //temp=C(H)*G
pnl_vect_complex_clone(G, tempg);

for (j = 1; j < n_points; j++)
{
    for (i = 0; i < 2 * M + 1; i++)
    {
        pnl_vect_complex_set(F, i, Cadd(cte, pnl_vect_complex_get(F, i))); //F
        pnl_vect_complex_set(G, i, Cadd(cte, pnl_vect_complex_get(G, i))); //G
    }
    F_diag_fi_kou(F, t, sigma, d, lambda, lambdap, lambdam, p, h, x, M, signe)
    toep(v, w, F, tempf); //
    pnl_vect_complex_clone(F, tempf); //F=temp=H.Df*F

    G_diag_fi_kou(G, t, sigma, d, lambda, lambdap, lambdam, p, h, x, M, signe)
    toep(v, w, G, tempg); //temp=C(H)*G
    pnl_vect_complex_clone(G, tempg); //G=temp=H.Dg*G

    cte = CRMul(CRadd(pnl_vect_complex_get(F, M), -1), -1);

}

k = CRMul(-1, CI);
a = fi_kou(n_points * t, sigma, d, lambda, lambdap, lambdam, p, CRMul(k, x), s

pnl_vect_complex_set(G, M, Cmul(Cadd(pnl_vect_complex_get(G, M), cte), a));

pnl_vect_complex_free(&v);
pnl_vect_complex_free(&w);
pnl_vect_complex_free(&tempf);
pnl_vect_complex_free(&tempg);
pnl_vect_complex_free(&F);

return pnl_vect_complex_get(G, M);

```

```
}
```

```
static int fft_kou_lookbackfixed(double s_maxmin, NumFunc_2 *P, double S0, double T)
{
    double pas, d, c, drift, nu, h_temp, x_maxmin, signe, beta, K;
    dcomplex h, res;
    PnlVectComplex *G;
    K = P->Par[0].Val.V_DOUBLE;
    beta = 0.5826;
    pas = T / n_points;
    //the width of the estimation's strip
    d = MIN(lambdam, lambdap);
    // the optimal h : h(M)//
    nu = 2;
    c = sigma * sigma / 2;
    h_temp = pow((M_PI * d) / (pas * c), 1.0 / (1 + nu)) * pow((double)M, -nu / 2);
    h = CRmul(CONE, h_temp);
    //martingale condition
    drift = r - divid - c + lambda * (1 - ((p * lambdam) / (lambdam - 1)) - (((1 - p) * lambdap) / (lambdap - 1)));
    G = pnl_vect_complex_create(2 * M + 1);

    //CALL
    if ((P->Compute) == &Call_OverSpot2)
    {
        s_maxmin = MAX(s_maxmin, K);
        s_maxmin = exp(-beta * sigma * sqrt(T / n_points)) * s_maxmin;
        signe = 1;
        x_maxmin = signe * log((s_maxmin / S0));
        res = estimation_kou(x_maxmin, G, pas, sigma, drift, lambda, lambdap, lambdam);
        res = CRsub(CRmul(res, exp(beta * sigma * sqrt(T / n_points))) * exp(-r * T));
    }
    //PUT
    if ((P->Compute) == &Put_OverSpot2)
    {
        s_maxmin = MIN(s_maxmin, K);
        s_maxmin = exp(beta * sigma * sqrt(T / n_points)) * s_maxmin;
        signe = -1;
        x_maxmin = signe * log((s_maxmin / S0));
        res = estimation_kou(x_maxmin, G, pas, sigma, drift, lambda, lambdap, lambdam);
        res = CRadd(CRmul(res, -S0 * exp(-beta * sigma * sqrt(T / n_points))) * exp(-r * T));
    }
}
```

```

    }
    *ptprice = Creal(res);

    pnl_vect_complex_free(&G);
    return OK;
}

int CALC(FFT_Kou_FixedLookback)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid;

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

    return fft_kou_lookbackfixed((ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[4].Val.V_P

}

static int CHK_OPT(FFT_Kou_FixedLookback)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "LookBackCallFixedEuro") == 0) || (strcmp((
        return OK;
    return WRONG;
}

#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Mod)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "ap_kou_lookbackfixed_fft";
        Met->Par[0].Val.V_PINT = 100;
        Met->Par[1].Val.V_LONG = 512;
    }
    return OK;
}

PricingMethod MET(FFT_Kou_FixedLookback) =
{
    "FFT_Kou_LookbackFixed",

```

```

    {"Number of discretization steps", LONG, {100}, ALLOW}, {"N truncation level",
    CALC(FFT_Kou_FixedLookback),
    {"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(FFT_Kou_FixedLookback),
    CHK_ok,
    MET(Init)
} ;

```