

Help

```

extern "C" {
#include "kou1d_lim.h"
}
#include<iostream>
#include<cmath>
#include"math/ap_kou_model/functions.h"

extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
    static int CHK_OPT(AP_Kou_Out)(void *Opt, void *Mod)
    {
        return NONACTIVE;
    }
    int CALC(AP_Kou_Out)(void *Opt, void *Mod, PricingMethod *Met)
    {
        return AVAILABLE_IN_FULL_PREMIA;
    }
#else
    static int Kou_Ap_Out(int b_type, double l, double rebate, double S0, NumFunc_
    {
        long double ksi, cst1, cst2, dcst1, dcst2, proba, proba2, dproba, temp, dpro
        long double x[9];
        int op_type = 0;
        long double h = 0.01;

        /*Call Case*/
        if ((P->Compute) == &Call)
            op_type = 0;
        else
            /*Put Case*/
            if ((P->Compute) == &Put)
                op_type = 1;

        ksi = p * lambdap / (lambdap - 1) + (1 - p) * lambdam / (lambdam + 1) - 1;
        double K = P->Par[0].Val.V_DOUBLE;

        if (b_type == 0) //down
        {
            x[0] = -((r - divid) - sigma * sigma / 2 - lambda * ksi);

```

```

x[1] = sigma;
x[2] = lambda;
x[3] = 1 - p;
x[4] = lambdam;
x[5] = lambdap;
x[6] = log(S0 / K);
x[7] = log(S0 / l);
x[8] = T;

if (op_type == 0) //call
{
    proba = psiM(x, T);
    cst1 = psiB(x, T);
    x[6] = log((S0 + h) / K);
    x[7] = log((S0 + h) / l);
    dproba = psiM(x, T);
    dcst1 = psiB(x, T);
    x[7] = T;
    dcst1 = 1 - psiVN(x) - dproba + dcst1;
    x[6] = log(S0 / K);
    cst1 = 1 - psiVN(x) - proba + cst1;
}
else//put
{
    proba = psiM(x, T);
    cst1 = psiB(x, T);
    x[6] = log((S0 + h) / K);
    x[7] = log((S0 + h) / l);
    dproba = psiM(x, T);
    dcst1 = psiB(x, T);
    x[7] = T;
    dcst1 = psiVN(x) - dcst1;
    x[6] = log(S0 / K);
    cst1 = psiVN(x) - cst1;
}
x[7] = log((S0 + h) / l);
dproba = rebateproba(x, r, T);
x[6] = log(S0 / K);
x[7] = log(S0 / l);
proba = rebateproba(x, r, T);
x[0] = -((r - divid) + sigma * sigma / 2 - lambda * ksi);

```

```

x[2] = lambda * (ksi + 1);
x[3] = p * lambdap / ((1 + ksi) * (lambdap - 1));
x[3] = 1 - x[3];
x[4] = lambdap - 1;
x[5] = lambdam + 1;
temp = x[4];
x[4] = x[5];
x[5] = temp;

if (op_type == 0) //call
{
    proba2 = psiM(x, T);
    cst2 = psiB(x, T);
    x[6] = log((S0 + h) / K);
    x[7] = log((S0 + h) / 1);
    dproba2 = psiM(x, T);
    dcst2 = psiB(x, T);
    x[7] = T;
    dcst2 = 1 - psiVN(x) - dproba2 + dcst2;
    x[6] = log(S0 / K);
    cst2 = 1 - psiVN(x) - proba2 + cst2;

    *ptPrice = S0 * exp(-divid * T) * cst2 - K * exp(-r * T) * cst1 + re
    dptPrice = (S0 + h) * exp(-divid * T) * dcst2 - K * exp(-r * T) * dc
    *ptDelta = (dptPrice - *ptPrice) / h;
}
else//put
{
    cst2 = psiB(x, T);
    x[6] = log((S0 + h) / K);
    x[7] = log((S0 + h) / 1);
    dcst2 = psiB(x, T);
    x[7] = T;
    dcst2 = psiVN(x) - dcst2;
    x[6] = log(S0 / K);
    cst2 = psiVN(x) - cst2;

    *ptPrice = K * exp(-r * T) * cst1 - S0 * exp(-divid * T) * cst2 + re
    dptPrice = K * exp(-r * T) * dcst1 - (S0 + h) * exp(-divid * T) * dc
    *ptDelta = (dptPrice - *ptPrice) / h;
}

```

```

    }
else//up
{
    x[0] = (r - divid) - sigma * sigma / 2 - lambda * ksi;
    x[1] = sigma;
    x[2] = lambda;
    x[3] = p;
    x[4] = lambdap;
    x[5] = lambdam;
    x[6] = log(K / S0);
    x[7] = log(1 / S0);
    x[8] = T;

    if (op_type == 0) //call
    {
        proba = psiM(x, T);
        cst1 = psiB(x, T);
        x[6] = log(K / (S0 + h));
        x[7] = log(1 / (S0 + h));
        dproba = psiM(x, T);
        dcst1 = psiB(x, T);
        x[7] = T;
        dcst1 = psiVN(x) - dcst1;
        x[6] = log(K / S0);
        cst1 = psiVN(x) - cst1;
    }
else//put
{
    proba = psiM(x, T);
    cst1 = psiB(x, T);
    x[6] = log(K / (S0 + h));
    x[7] = log(1 / (S0 + h));
    dproba = psiM(x, T);
    dcst1 = psiB(x, T);
    x[7] = T;
    dcst1 = 1 - psiVN(x) - dproba + dcst1;
    x[6] = log(K / S0);
    cst1 = 1 - psiVN(x) - proba + cst1;
}
x[7] = log(1 / (S0 + h));
dproba = rebateproba(x, r, T);

```

```

x[6] = log(K / S0);
x[7] = log(1 / S0);
proba = rebateproba(x, r, T);
x[0] = (r - divid) + sigma * sigma / 2 - lambda * ksi;
x[2] = lambda * (ksi + 1);
x[3] = p * lambdap / ((1 + ksi) * (lambdap - 1));
x[4] = lambdap - 1;
x[5] = lambdam + 1;

if (op_type == 0) //call
{
    cst2 = psiB(x, T);
    x[6] = log(K / (S0 + h));
    x[7] = log(1 / (S0 + h));
    dcst2 = psiB(x, T);
    x[7] = T;
    dcst2 = psiVN(x) - dcst2;
    x[6] = log(K / S0);
    cst2 = psiVN(x) - cst2;

    *ptPrice = S0 * exp(-divid * T) * cst2 - K * exp(-r * T) * cst1 + re
    dptPrice = (S0 + h) * exp(-divid * T) * dcst2 - K * exp(-r * T) * dc
    *ptDelta = (dptPrice - *ptPrice) / h;
}
else//put
{
    proba2 = psiM(x, T);
    cst2 = psiB(x, T);
    x[6] = log(K / (S0 + h));
    x[7] = log(1 / (S0 + h));
    dproba2 = psiM(x, T);
    dcst2 = psiB(x, T);
    x[7] = T;
    dcst2 = 1 - psiVN(x) - dproba2 + dcst2;
    x[6] = log(K / S0);
    cst2 = 1 - psiVN(x) - proba2 + cst2;

    *ptPrice = K * exp(-r * T) * cst1 - S0 * exp(-divid * T) * cst2 + re
    dptPrice = K * exp(-r * T) * dcst1 - (S0 + h) * exp(-divid * T) * dc
    *ptDelta = (dptPrice - *ptPrice) / h;
}

```

```

    }

    return OK;
}

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

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
    limit = ((ptOpt->Limit.Val.V_NUMFUNC_1)->Compute)((ptOpt->Limit.Val.V_NUMFUN
    rebate = ((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compute)((ptOpt->Rebate.Val.V_NUM

    if ((ptOpt->DownOrUp).Val.V_BOOL == DOWN)
        upordown = 0;
    else upordown = 1;

    return Kou_Ap_Out(upordown, limit, rebate, ptMod->S0.Val.V_PDOUBLE, ptOpt->
}

static int CHK_OPT(AP_Kou_Out)(void *Opt, void *Mod)
{
    Option *ptOpt = (Option *)Opt;
    TYPEOPT *opt = (TYPEOPT *) (ptOpt->TypeOpt);

    /* if ((opt->RebOrNo).Val.V_BOOL==NOREBATE)*/
    if ((opt->OutOrIn).Val.V_BOOL == OUT)
        if ((opt->EuOrAm).Val.V_BOOL == EURO)
            if ((opt->Parisian).Val.V_BOOL == FALSE)
                return OK;

    return WRONG;
}

#endif //PremiaCurrentVersion

```

```
static int MET(Init)(PricingMethod *Met, Option *Mod)
{
    return OK;
}

PricingMethod MET(AP_Kou_Out) =
{
    "AP_Kou_Barrier_Out",
    {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_Kou_Out),
    {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID}, {" ", P
    CHK_OPT(AP_Kou_Out),
    CHK_ok,
    MET(Init)
} ;
}
```