

[Help](#)

```

#include <stdlib.h>
#include "kould_std.h"
#include "math/wienerhopf.h"

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

/*////////////////////////////////////*/
static int wh_kou_amerput(int ifCall, double Spot, double sigma, double lambda,
                        double r, double divid,
                        double T, double h, double Strike1,
                        double er, long int step,
                        double *ptprice, double *ptdelta)
{

    double cp, cm, ptprice1, ptdelta1, mu, qu, omega, sig2, lm, lp;

    lp = lambdam;
    lm = -lambdap;

    if (ifCall == 0)
    {
        omega = lm < -2. ? 2. : (-lm + 1.) / 2.;
    }
    else
    {
        omega = lp > 1. ? -1. : -lp / 2.;
    }

    cp = (1 - P) * lambda;
    cm = P * lambda;

```

```

sig2 = sigma * sigma;

mu = r - divid + cp / (lp + 1.0) + cm / (lm + 1.0) - sig2 / 2.0;

qu = r - mu * omega - sig2 * omega * omega / 2 + cp + cm - cp * lp / (lp + omega)

fastwienerhopfamerican(4, mu, qu, omega, ifCall, Spot, lm, lp,
                        2.0, sigma, cm, cp, r, divid,
                        T, h, Strike1,
                        er, step, &ptprice1, &ptdelta1);

//Price
*ptprice = ptprice1;
//Delta
*ptdelta = ptdelta1;
return OK;
}

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

    NumFunc_1 *p;
    int res;

    int ifCall;

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

    p = ptOpt->PayOff.Val.V_NUMFUNC_1;
    strike = p->Par[0].Val.V_DOUBLE;
    spot = ptMod->S0.Val.V_DOUBLE;
    ifCall = ((p->Compute) == &Call);

```

```

    res = wh_kou_amerput(ifCall, spot, ptMod->Sigma.Val.V_PDDOUBLE, ptMod->Lambda.Val.V_PDDOUBLE,
                        r, divid,
                        ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, Met->Par[0].Val.V_DOUBLE, Met->Par[2].Val.V_INT2,
                        Met->Par[0].Val.V_DOUBLE, Met->Par[2].Val.V_INT2,
                        Met->Res[0].Val.V_DOUBLE, Met->Res[1].Val.V_DOUBLE);

    return res;
}

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

    if ((strcmp(((Option *)Opt)->Name, "PutAmer") == 0) || (strcmp(((Option *)Opt)->Name, "CallAmer") == 0))
        return OK;

    return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    static int first = 1;

    if (first)
    {
        Met->Par[0].Val.V_PDDOUBLE = 2.0;
        Met->Par[1].Val.V_PDDOUBLE = 0.01;
        Met->Par[2].Val.V_INT2 = 600;

        first = 0;
    }

    return OK;
}

PricingMethod MET(AP_fastwhamer_kou) =
{

```

```
"AP_FastWH_Kou",
{ {"Scale of logprice range", DOUBLE, {100}, ALLOW},
  {"Space Discretization Step", DOUBLE, {500}, ALLOW},
  {"TimeStepNumber", INT2, {100}, ALLOW},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CALC(AP_fastwhamer_kou),
{ {"Price", DOUBLE, {100}, FORBID},
  {"Delta", DOUBLE, {100}, FORBID},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CHK_OPT(AP_fastwhamer_kou),
CHK_split,
MET(Init)
};
```