

Help

```
#include <stdlib.h>
#include "rskould_lim.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_fft.h"
#include "math/wienerhopf_rs.h"

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

static char *infilename;

static int wh_rskou_bar(int am, int upordown, int ifCall, double Spot,
                        double T, double h, double Strike1,
                        double bar, double rebate,
                        double er, long int step, int n_state,
                        double *ptprice, double *ptdelta)
{
    PnlVect *divi, *rr, *lambda, *pp, *lambdap, *lambdam, *cm, *cp, *strike, *sigm
    PnlVect *prices, *deltas;
    double eps;
    PnlMat *lam;
    int res, i, nstates;
    double tomega, omegas, sig2;

    eps = 1.0e-7; // accuracy of iterations

    res = readparamskou_rs(&nstates, &rr, &divi, &sigmas, &lambdam, &lambdap, &lam

    if (!res)
    {
        printf("An error occured while reading file!\ n");
    }
}
```

```

    *ptprice = 0.;
    *ptdelta = 0.;
    return OK;
}

mu = pnl_vect_create(nstates + 1);
qu = pnl_vect_create(nstates + 1);
cp = pnl_vect_create(nstates + 1);
cm = pnl_vect_create(nstates + 1);
strike = pnl_vect_create(nstates + 1);
rebates = pnl_vect_create(nstates + 1);
prices = pnl_vect_create(nstates + 1);
deltas = pnl_vect_create(nstates + 1);

for (i = 0; i < nstates; i++) LET(strike, i) = Strike1;

if (upordown == 0)
{
    omegas = 2.0;
}
else
{
    omegas = -1.0;
}

for (i = 0; i < nstates; i++)
{
    LET(rr, i) = log(1. + GET(rr, i) / 100.);
    LET(divi, i) = log(1. + GET(divi, i) / 100.);
    LET(rebates, i) = rebate;

    if (upordown == 0)
    {
        tomega = GET(lambdam, i) < -2. ? 2. : (-GET(lambdam, i) + 1.) / 2.;
        omegas = omegas > tomega ? tomega : omegas;
    }
    else
    {
        tomega = GET(lambdap, i) > 1. ? -1. : -GET(lambdap, i) / 2.;
        omegas = omegas < tomega ? tomega : omegas;
    }
}

```

```

    LET(cp, i) = (1 - GET(pp, i)) * GET(lambda, i);
    LET(cm, i) = GET(pp, i) * GET(lambda, i);
    sig2 = GET(sigmas, i) * GET(sigmas, i);

    LET(mu, i) = GET(rr, i) - GET(divi, i) + GET(cp, i) / (GET(lambdap, i) + 1);

    LET(qu, i) = GET(rr, i) - GET(mu, i) * omegas - sig2 * omegas * omegas / 2;
}

res = fastwienerhopf_rs(4, nstates, mu, qu, omegas, 1, upordown, ifCall, Spot,
                        T, h, strike, bar, rebates, er, step, eps, prices, del

//Price
*ptprice = GET(prices, n_state - 1);
//Delta
*ptdelta = GET(deltas, n_state - 1);

// Memory desallocation
pnl_vect_free(&mu);
pnl_vect_free(&qu);
pnl_vect_free(&prices);
pnl_vect_free(&deltas);
pnl_vect_free(&rr);
pnl_vect_free(&divi);
pnl_vect_free(&sigmas);
pnl_vect_free(&lambdap);
pnl_vect_free(&lambdam);
pnl_vect_free(&cp);
pnl_vect_free(&cm);
pnl_vect_free(&lambda);
pnl_vect_free(&pp);
pnl_vect_free(&strike);
pnl_vect_free(&rebates);

pnl_mat_free(&lam);

return OK;
}

//=====

```

```

int CALC(AP_fastwhdownout_rskou)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double limit, strike, spot, rebate;

    NumFunc_1 *p;
    int res;
    int upordown;
    int ifCall;

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

    infilename = ptMod->Transition_probabilities.Val.V_FILENAME;

    rebate = ((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compute)((ptOpt->Rebate.Val.V_NUMFUNC_1)->

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

    res = wh_rskou_bar(ptOpt->EuOrAm.Val.V_BOOL, upordown, ifCall, spot,
        ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, Met->Par[0].Val.V_DOUBLE,
        limit, rebate,
        Met->Par[0].Val.V_DOUBLE, Met->Par[2].Val.V_INT2, Met->Par[2].Val.V_DOUBLE,
        &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE));

    return res;
}

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

    // return NONACTIVE;

```

```

    if ((opt->OutOrIn).Val.V_BOOL == OUT)
        if ((opt->Parisian).Val.V_BOOL == FALSE)
            if ((opt->EuOrAm).Val.V_BOOL == EURO)
                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_PDOUBLE = 2.0;
        Met->Par[1].Val.V_PDOUBLE = 0.001;
        Met->Par[2].Val.V_INT2 = 10;
        Met->Par[3].Val.V_INT = 1;
        first = 0;
    }
    return OK;
}

PricingMethod MET(AP_fastwhdownout_rskou) =
{
    "AP_FastWHBar_RSKOU",
    { {"Scale of logprice range", DOUBLE, {100}, ALLOW},
      {"Space Discretization Step", DOUBLE, {500}, ALLOW},
      {"TimeStepNumber", INT2, {100}, ALLOW},
      {"Output state number", INT, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(AP_fastwhdownout_rskou),
    { {"Price of chosen state", DOUBLE, {100}, FORBID},
      {"Delta of chosen state", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_fastwhdownout_rskou),
    CHK_split,

```

```
    MET(Init)  
};
```