

## Help

```

#include "hes1d_lim.h"
#include <stdlib.h>
#include "pnl/pnl_vector.h"
#include "pnl/pnl_fft.h"
#include "math/wienerhopf_rs.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2011+2) //The "#els
static int CHK_OPT(AP_fastwhbar_hes)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_fastwhbar_hes)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

int wh_hes_bar(int upordown, int ifCall, double Spot,
               double r, double divi, double v0, double kappa, double theta, dou
               double T, double Strike,
               double bar, double Rebate, double er, double h, long int step,
               int nstates, double ver,
               double *ptprice, double *ptdelta)
{
    PnlVect *shftp, *shftm, *volh, *svar, *mu, *qu;
    PnlVect *prices, *deltas;
    double omh, kah, tok, rok, mu0, rov, var, mui;
    double vmin, vmax, vh;
    int k0;
    PnlMat *lam;
    int i, j;
    double omegas;
    double lambdap, lambdam, cm, cp;

    mu = pnl_vect_create(nstates + 1);
    qu = pnl_vect_create(nstates + 1);
    shftp = pnl_vect_create(nstates + 1);

```

```

shftm = pnl_vect_create(nstates + 1);
volh = pnl_vect_create(nstates + 1);
svar = pnl_vect_create(nstates + 1);
prices = pnl_vect_create(nstates + 1);
deltas = pnl_vect_create(nstates + 1);
lam = pnl_mat_create(nstates + 1, nstates + 1);

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

//to change below

v0 = pow(v0, 0.5);
vmax = 6.*v0 * ver;
vmin = v0 / 8. / ver;
vh = pow(vmax / vmin, 1. / (nstates - 1));

k0 = (int)ceil(log(v0 / vmin) / log(vh)); //warning k0<nstates

for (i = 0; i < nstates; i++)
{
    LET(svar, i) = v0 * pow(vh, (i - k0 + 1));
}

for (i = 0; i < nstates - 1; i++)
{
    LET(volh, i) = GET(svar, i + 1) - GET(svar, i);
}
LET(volh, nstates - 1) = 0;

omh = 0.5 * omega;
omh = -omh * omh;
kah = kappa / 2.;

```

```

tok = theta - omega * omega / kappa / 4;
rok = rho * kappa / omega;
rov = 2 * rho / omega;
mu0 = r - rok * tok - divi; //!!!
//////////not corrected transition matrix - always positive
for (i = 0; i < nstates; i++)
{
    for (j = 0; j < nstates; j++)
    {
        MLET(lam, i, j) = 0.0;
    }
}

for (i = 1; i < nstates - 1; i++)
{
    var = GET(svar, i) * GET(svar, i);
    mui = kah * (tok - var) / GET(svar, i);
    if (tok > var)
    {
        MLET(lam, i, i - 1) = (-omh - mui * GET(volh, i)) / (GET(volh, i - 1)
        if (MGET(lam, i, i - 1) >= 0)
        {
            MLET(lam, i, i + 1) = (-omh + mui * GET(volh, i - 1)) / (GET(volh,
        }
        else
        {
            MLET(lam, i, i) = omh / (GET(volh, i - 1) + GET(volh, i));
            MLET(lam, i, i - 1) = -MGET(lam, i, i) / GET(volh, i - 1);
            MLET(lam, i, i + 1) = (-MGET(lam, i, i) + mui) / GET(volh, i);
        }
    }
    else
    {
        MLET(lam, i, i + 1) = (-omh + mui * GET(volh, i - 1)) / (GET(volh, i -
        if (MGET(lam, i, i + 1) >= 0)
        {
            MLET(lam, i, i - 1) = (-omh - mui * GET(volh, i)) / (GET(volh, i -
        }
        else
        {
            MLET(lam, i, i) = omh / (GET(volh, i - 1) + GET(volh, i));

```

```

        MLET(lam, i, i - 1) = (-MGET(lam, i, i) - mui) / GET(volh, i - 1);
        MLET(lam, i, i + 1) = -MGET(lam, i, i) / GET(volh, i);
    }
}
MLET(lam, i, i) = -MGET(lam, i, i - 1) - MGET(lam, i, i + 1);
}
var = GET(svar, 0) * GET(svar, 0);
MLET(lam, 0, 0) = omh / GET(volh, 0);
if (tok > var)
{
    MLET(lam, 0, 1) = (-MGET(lam, 0, 0) + kah * (tok - var) / GET(svar, 0)) /
    MLET(lam, 0, 0) = -MGET(lam, 0, 1);
}
else
{
    MLET(lam, 0, 1) = -MGET(lam, 0, 0) / GET(volh, 0);
    MLET(lam, 0, 0) = -MGET(lam, 0, 1);
}

var = GET(svar, nstates - 1) * GET(svar, nstates - 1);
MLET(lam, nstates - 1, nstates - 1) = omh / (GET(volh, nstates - 1) + GET(volh,
if (tok > var)
{
    MLET(lam, nstates - 1, nstates - 2) = -MGET(lam, nstates - 1, nstates - 1)
    MLET(lam, nstates - 1, nstates - 1) = -MGET(lam, nstates - 1, nstates - 2)
}
else
{
    MLET(lam, nstates - 1, nstates - 2) = (-MGET(lam, nstates - 1, nstates - 1)
    MLET(lam, nstates - 1, nstates - 1) = -MGET(lam, nstates - 1, nstates - 2)
}
////////////////////end of transition matrix////////////////
LET(shftm, 0) = 0;
for (i = 0; i < nstates; i++)
{
    LET(shftp, i) = rov * GET(svar, i) * GET(volh, i);
    if (i > 0)
    {
        LET(shftm, i) = rov * GET(svar, i) * GET(volh, i - 1);
    }
    LET(mu, i) = mu0 - (0.5 - rok) * GET(svar, i) * GET(svar, i);

```

```

    LET(svar, i) = (1 - rho * rho) * GET(svar, i) * GET(svar, i);
    LET(qu, i) = r - GET(mu, i) * omegas - GET(svar, i) * omegas * omegas / 2.
    LET(mu, i) = GET(mu, i) + omegas * GET(svar, i);
    lambdap = 5.0;
    lambdam = -5.0;
    cm = 0.0;
    cp = 0.0;
}

////////////////////////////////////////ok
fastwienerhopf_hs(6, nstates, mu, qu, omegas, 1, upordown, ifCall,
    Spot, lambdam, lambdap, svar, shftm, shftp, cm,
    cp, r, divi, lam, T, h, Strike, bar, Rebate, er,
    step, prices, deltas);

//Price
*ptprice = GET(prices, k0 - 1);
//Delta
*ptdelta = GET(deltas, k0 - 1);

// Memory deallocation
pnl_vect_free(&mu);
pnl_vect_free(&qu);
pnl_vect_free(&prices);
pnl_vect_free(&deltas);
pnl_vect_free(&shftm);
pnl_vect_free(&shftp);
pnl_vect_free(&svar);
pnl_vect_free(&volh);
pnl_mat_free(&lam);

return OK;
}

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

```

```

double r, divid;
int upordown;
int res;
int ifCall;
NumFunc_1 *p;

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_NUMFUNC_1)->Compute);
p = ptOpt->PayOff.Val.V_NUMFUNC_1;
strike = p->Par[0].Val.V_DOUBLE;
spot = ptMod->S0.Val.V_DOUBLE;
ifCall = ((p->Compute) == &Call);

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

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

res = wh_hes_bar(upordown, ifCall, spot, r,
                 divid, ptMod->Sigma0.Val.V_PDOUBLE,
                 , ptMod->MeanReversion.Val.V_PDOUBLE,
                 ptMod->LongRunVariance.Val.V_PDOUBLE,
                 ptMod->Sigma.Val.V_PDOUBLE,
                 ptMod->Rho.Val.V_PDOUBLE,
                 ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                 strike, limit, rebate,
                 Met->Par[0].Val.V_DOUBLE, Met->Par[1].Val.V_DOUBLE, Met->Par[2].Val.V_DOUBLE,
                 , Met->Par[3].Val.V_INT2, Met->Par[4].Val.V_DOUBLE,
                 &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE));
//double er, double h, long int step, int nstates, double ver,
return res;
}

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

```

```

    if ((opt->OutOrIn).Val.V_BOOL == OUT &&
        (opt->Parisian).Val.V_BOOL == FALSE &&
        (opt->EuOrAm).Val.V_BOOL == EURO)
        return OK;

    return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "AP_fastwhbar_hes";
        Met->Par[0].Val.V_DOUBLE = 2.0;
        Met->Par[1].Val.V_DOUBLE = 0.01;
        Met->Par[2].Val.V_INT2 = 400;
        Met->Par[3].Val.V_INT2 = 20;
        Met->Par[4].Val.V_DOUBLE = 1;
    }
    return OK;
}

PricingMethod MET(AP_fastwhbar_hes) =
{
    "AP_FastWHBar_HES",
    { {"Scale of logprice range", DOUBLE, {100}, ALLOW},
      {"Space Discretization Step", DOUBLE, {500}, ALLOW},
      {"TimeStepNumber", INT2, {100}, ALLOW},
      {"Number of the states", INT2, {100}, ALLOW},
      {"Scale of volatility range", DOUBLE, {500}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(AP_fastwhbar_hes),
    { {"Price ", DOUBLE, {100}, FORBID},
      {"Delta ", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_fastwhbar_hes),

```

```
    CHK_split,  
    MET(Init)  
};
```