

[Help](#)

```

#include "hbw4d_std.h"
#include <math.h>
#include "pnl/pnl_vector.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_integration.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2013+2) //The "#els
static int CHK_OPT(AP_HHW4D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_HHW4D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double u0, kappa0, vbar0, rho120, rho130, rho140, rho340, gammav0, rho230
dcomplex fd0, fg0;

void static funcB(double u, dcomplex *result)
{
    *result = CRMul(CI, u);
}
void static funcd(double u, double kappa, double gamma, double rho12, dcomplex *
{
    if (kappa == 0.0 || gamma == 0.0) *result = CZERO;
    else *result = Csqrt(Csub(Cpow_real(CRsub(CRMul(CI, rho12 * gamma * u), kap
}
void static funcg(double u, double kappa, double gamma, double rho12, dcomplex f
{
    if (Cimag(fd) == 0.0 && Creal(fd) == 0.0) *result = CONE;
    else *result = Cdiv(Csub(RCsub(kappa, CRMul(gamma * rho12 * u, CI)), fd), C
}
void static funcC(double u, dcomplex fd, dcomplex fg, double tau, double kappa,

```

```

{
    *result = Cmul(Cdiv(RCsub(1., Cexp(RCmul(-tau, fd))), RCmul(gamma * gamma, RCs
}
void static funcC1(double kappa, double gamma, double *result)
{
    if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
    else *result = gamma * gamma * (1. - exp(-kappa)) / (4.*kappa);
}
void static funcLambda(double kappa, double v0, double gamma, double *result)
{
    if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
    else *result = 4.*kappa * v0 * exp(-kappa) / (gamma * gamma * (1. - exp(-kappa)
}
void static funcD1(double kappa, double vbar, double gamma, double *result)
{
    if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
    else *result = 4.*kappa * vbar / (gamma * gamma);
}
void static fLambda1(double fC1, double fLambda, double fD1, double *result)
{
    if (fD1 == 0.0 || fLambda == 0.0) *result = 0.0;
    else *result = sqrt(fC1 * (fLambda - 1.) + fC1 * fD1 + fC1 * fD1 / (2.*(fD1 +
}
void static funcbeta1(double kappa, double vbar, double gamma, double *result)
{
    if (vbar - gamma * gamma / 8. / kappa < 0.0)
    {
        *result = 0.0;
        printf("Invalid parameter for vbar, gammav, kappa!\ n");
    }
    else if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
    else *result = sqrt(vbar - gamma * gamma / 8. / kappa);
}
void static funcbeta2(double v0, double beta1, double *result)
{
    *result = sqrt(v0) - beta1;
}
void static funcbeta3(double beta1, double beta2, double Lambda1, double *result)
{
    if (beta2 == 0.0 || (Lambda1 - beta1) == 0.0) *result = 0.0;
    else *result = -log((Lambda1 - beta1) / beta2);
}

```

```

}
static double intgf1r(double x, void *p)
{
    dcomplex fC;
    funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
    return (kappa0 * vbar0 + rho230 * gammav0 * etad0 * (beta10 + beta20 * exp(-be
}
static double intgf2r(double x, void *p)
{
    dcomplex fC;
    funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
    return rho230 * etad0 * gammav0 * (beta10 + beta20 * exp(-beta30 * x)) * (exp(
}
static double intgf3r(double x, void *p)
{
    dcomplex fC;
    funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
    return rho240 * gammav0 * etaf0 * (beta10 + beta20 * exp(-beta30 * x)) * (exp(
}
static double intgf1i(double x, void *p)
{
    dcomplex fC;
    funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
    return (kappa0 * vbar0 + rho230 * gammav0 * etad0 * (beta10 + beta20 * exp(-be
}
static double intgf2i(double x, void *p)
{
    dcomplex fC;
    funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
    return rho230 * etad0 * gammav0 * (beta10 + beta20 * exp(-beta30 * x)) * (exp(
}
static double intgf3i(double x, void *p)
{
    dcomplex fC;
    funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
    return rho240 * gammav0 * etaf0 * (beta10 + beta20 * exp(-beta30 * x)) * (exp(
}
static double intgfzeta(double x, void *p)
{
    return (rho130 * etad0 * (exp(-lambdad0 * (T0 - x)) - 1.) / lambdad0 - rho140
}

```

```

void static funcA(double u, double v0, double kappa, double vbar, double gammav,
{
    double intg1r = 0., intg2r = 0., intg3r = 0., intg1i = 0., intg2i = 0., intg3i
    double fC1, fLambda, fD1, Lambda1, beta1, beta2, beta3;
    dcomplex fd, fg;
    PnlFunc func1r, func2r, func3r, func1i, func2i, func3i, funczeta;
    int n = 50;
    funcd(u, kappa, gammav, rho12, &fd);
    funcg(u, kappa, gammav, rho12, fd, &fg);
    funcbeta1(kappa, vbar, gammav, &beta1);
    funcbeta2(v0, beta1, &beta2);
    funcC1(kappa, gammav, &fC1);
    funcLambda(kappa, v0, gammav, &fLambda);
    funcD1(kappa, vbar, gammav, &fD1);
    fLambda1(fC1, fLambda, fD1, &Lambda1);
    funcbeta3(beta1, beta2, Lambda1, &beta3);
    fd0 = fd;
    fg0 = fg;
    u0 = u;
    kappa0 = kappa;
    vbar0 = vbar;
    rho120 = rho12;
    rho130 = rho13;
    rho140 = rho14;
    rho340 = rho34;
    gammav0 = gammav;
    rho230 = rho23;
    etad0 = etad;
    lambdad0 = lambdad;
    rho240 = rho24;
    etaf0 = etaf;
    lambdaf0 = lambdaf;
    beta10 = beta1;
    beta20 = beta2;
    beta30 = beta3;
    T0 = T;
    func1r.F = intgf1r;
    func2r.F = intgf2r;
    func3r.F = intgf3r;
    func1i.F = intgf1i;
    func2i.F = intgf2i;

```

```

func3i.F = intgf3i;
funczeta.F = intgfzeta;
intg1r = pnl_integration(&func1r, 0.0, T, n, "simpson");
intg2r = pnl_integration(&func2r, 0.0, T, n, "simpson");
intg3r = pnl_integration(&func3r, 0.0, T, n, "simpson");
intg1i = pnl_integration(&func1i, 0.0, T, n, "simpson");
intg2i = pnl_integration(&func2i, 0.0, T, n, "simpson");
intg3i = pnl_integration(&func3i, 0.0, T, n, "simpson");
intgzeta = pnl_integration(&funczeta, 0.0, T, n, "simpson");
*result = Cadd(RCadd(intg1r + u * intg2i - u * intg3i, RCmul(intg1i - u * intg
}
void static chf(double u, double v0, double kappa, double vbar, double gammav, d
{
    dcomplex fb, fd, fg, fc, fa, expon;
    funcB(u, &fb);
    funcd(u, kappa, gammav, rho12, &fd);
    funcg(u, kappa, gammav, rho12, fd, &fg);
    funcC(u, fd, fg, T, kappa, gammav, rho12, &fc);
    funcA(u, v0, kappa, vbar, gammav, rho12, rho13, rho14, rho23, rho24, rho34, et
    expon = Cadd(Cadd(fa, RCmul(v0, fc)) , RCmul(-0.1 * delta_n * sqrt(T), fb));
    *result = Cexp(expon);
}

//COSINE method to calculate inverse fourier integral
double cosine_function_xi(double d, double c, double dma, double cma, double fnpi)
{
    double res = 1. / (1 + fnpi * fnpi) * ((cos(dma) + fnpi * sin(dma)) * exp(d) -
    return res;
}
double cosine_function_psi(double d, double c, double dma, double cma, double fn
{
    double res = (fnpi == 0.) ? (d - c) : (sin(dma) - sin(cma)) / fnpi;
    return res;
}
double cosine_Vk(double K, double a, double b, double fnpi)
{
    double cma = -a * fnpi;
    double bma = (b - a) * fnpi;
    double result = 2. / (b - a) * K * (cosine_function_xi(b, 0, bma, cma, fnpi) -
    return result;
}

```

```

void static cosine_bound(double L, double csi0, double pd0, double pf0, double d
{
    double c1, zeta, fC1, fLambda, Lambda1, fD1, beta1, beta2, beta3;
    funcbeta1(kappa, vbar, gammav, &beta1);
    funcbeta2(v0, beta1, &beta2);
    funcC1(kappa, gammav, &fC1);
    funcLambda(kappa, v0, gammav, &fLambda);
    funcD1(kappa, vbar, gammav, &fD1);
    fLambda1(fC1, fLambda, fD1, &Lambda1);
    funcbeta3(beta1, beta2, Lambda1, &beta3);

    zeta = etad * etad * log(exp(-lambdad * T)) * pow(lambdad, -0.3e1) / 0.2e1 + r

    c1 = csi0 * exp(-pf0 * T) / exp(-pd0 * T) + v0 / 2. / kappa * (exp(-kappa * T)

    *a = c1 - L + (-0.1 * delta_n * sqrt(T));
    *b = c1 + L + (-0.1 * delta_n * sqrt(T));
}

static int ap_hhw4d(NumFunc_1 *p, double T, PnlVect *r, PnlVect *x0, double vbar
{
    double csi0, v0;
    double gammav, etad, etaf;
    double kappa, lambdad, lambdaf;
    double rho12, rho13, rho14, rho23, rho24, rho34;
    double pd0, pf0;
    double delta_n;
    double rd, rf;

    double sum = 0, a, b;
    double K;
    double invbma = 0.0;
    double fnpi = 0.0;
    dcomplex phi = CZERO;
    int i;
    int N;

    //-----Initialisation of variable

```

```

csi0 = GET(x0, 0);
v0 = GET(x0, 1);

rd = GET(r, 0);
rf = GET(r, 1);

pd0 = exp(-rd * T);
pf0 = exp(-rf * T);

kappa = GET(kappa_v, 0);
lambdad = GET(kappa_v, 1);
lambdaf = GET(kappa_v, 2);

gammav = GET(sigma, 0);
etad = GET(sigma, 1);
etaf = GET(sigma, 2);

rho12 = GET(rho, 0);
rho13 = GET(rho, 1);
rho14 = GET(rho, 2);
rho23 = GET(rho, 3);
rho24 = GET(rho, 4);
rho34 = GET(rho, 5);

//Strike computation
delta_n = p->Par[0].Val.V_PDDOUBLE;
K = csi0 * pf0 / pd0 * exp(0.1 * delta_n * sqrt(T));

N = pow(2, 9.);
cosine_bound(10, csi0, rd, rf, delta_n, v0, kappa, vbar, gammav, etad, lambdad, rf, etaf, lambdaf, rho12, rho13, rho14, rho23, rho24, rho34);
invbma = M_PI / (b - a);
chf(fnpi, v0, kappa, vbar, gammav, rd, etad, lambdad, rf, etaf, lambdaf, rho12, rho13, rho14, rho23, rho24, rho34);
sum = 0.5 * phi.r * cosine_Vk(K, a, b, fnpi);
for (i = 1; i < N; i++)
{
    fnpi += invbma;
    chf(fnpi, v0, kappa, vbar, gammav, rd, etad, lambdad, rf, etaf, lambdaf, rho12, rho13, rho14, rho23, rho24, rho34);
    sum += (phi.r * cos(fnpi * (-a)) - phi.i * sin(fnpi * (-a))) * cosine_Vk(K, a, b, fnpi);
}
*ptprice = sum * exp(-rd * T);
return OK;

```

```

}

int CALC(AP_HHW4D)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return ap_hhw4d(ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Maturity.Val.V_DATE-
                    ptMod->x0.Val.V_PNLVECT, ptMod->MeanReversion.Val.V_DOUBLE,
                    ptMod->kappa.Val.V_PNLVECT,
                    ptMod->sigma.Val.V_PNLVECT,
                    ptMod->rho.Val.V_PNLVECT,
                    &(Met->Res[0].Val.V_DOUBLE)
    );
}

static int CHK_OPT(AP_HHW4D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "CallFX") == 0))
        return OK;
    return WRONG;
}

#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    //int type_generator;
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "ap_grzlek";
    }

    return OK;
}

PricingMethod MET(AP_HHW4D) =
{
    "AP_HHW4D",

```



```
{{" ", PREMIA_NULLTYPE, {0}, FORBID}},  
CALC(AP_HHW4D),  
{ {"Price", DOUBLE, {100}, FORBID},  
  {" ", PREMIA_NULLTYPE, {0}, FORBID}  
},  
CHK_OPT(AP_HHW4D),  
CHK_ok,  
MET(Init)  
};
```