

[Help](#)

```
#include "
href../../../../mod/hesvasicek1d/hesvasicek1d_std/hesvasicek1d_std_h_src.pdfhesvasi
#include "
href../../../../common/enums_h_src.pdfenums.h"
#include "
href../../../../common/error_msg_h_src.pdferror_msg.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"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2015+2) //The "#els
static int CHK_OPT(AP_FOURIERCOSINE_HESVAS)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_FOURIERCOSINE_HESVAS)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

void static funcD1(dcomplex u, double kv, double sigmav, double rho12, dcomplex
{
    if (kv == 0.0 || sigmav == 0.0) *result = CZERO;
    else *result = Csqrt(Csub(Cpow_real(CRsub(RCmul(sigmav * rho12, Cmul(CI, u))),
}
void static funcg(dcomplex u, double kv, double sigmav, double rho12, dcomplex D
{
    if (Cimag(D1) == 0.0 && Creal(D1) == 0.0) *result = CZERO;
    else *result = Cdiv(Csub(RCsub(kv, RCmul(sigmav * rho12, Cmul(CI, u))), D1), C
}
void static funcBx(dcomplex u, dcomplex *result)
{
    *result = Cmul(u, CI);
}
void static funcBr(dcomplex u, double tau, double kappar, double sigmar, dcomple
{
```

```

    if (sigmar == 0.0)
    {
        if (kappar == 0.0) *result = RCmul(tau, CRsub(Cmul(u, CI), 1.0));
        else *result = RCmul((1.0 - exp(-kappar * tau)) / kappar, CRsub(Cmul(u, CI), 1.0));
    }
    else *result = RCmul((1. - exp(-tau * kappar)) / kappar, CRsub(Cmul(u, CI), 1.0));
}

void static funcBv(dcomplex u, double tau, double kappav, double sigmav, double *result)
{
    if (sigmav == 0.0)
    {
        if (kappav == 0.0) *result = RCmul(-tau / 2.0, Cadd(Cmul(u, u), Cmul(u, CI)));
        else *result = RCmul((exp(-kappav * tau) - 1.0) / (2.0 * kappav), Cadd(Cmul(u, u), Cmul(u, CI)));
    }
    else *result = Cmul(Cdiv(RCsub(1., Cexp(RCmul(-tau, D1))) , RCmul(sigmav * tau, D1)), RCmul(sigmav * tau, D1));
}

void static funcC(double t, double speed, double variance, double *result)
{
    if (speed == 0.0 || variance == 0.0) *result = 0.0;
    else *result = variance * variance * (1. - exp(-speed * t)) / (4.*speed);
}

void static funcLambda(double t, double speed, double initial, double variance, double *result)
{
    if (speed == 0.0 || variance == 0.0) *result = 0.0;
    else *result = 4.*speed * initial * exp(-speed * t) / (variance * variance);
}

void static funcD(double speed, double mean, double variance, double *result)
{
    if (speed == 0.0 || variance == 0.0) *result = 0.0;
    else *result = 4.*speed * mean / (variance * variance);
}

void static Lambda1(double t, double C, double Lambda, double D, double *result)
{
    if (D == 0.0 || Lambda == 0.0) *result = 0.0;
    else *result = sqrt(C * (Lambda - 1.) + C * D + C * D / (2.*(D + Lambda)));
}

void static contA(double mean, double speed, double variance, double *result)
{
    if (speed == 0.0 || variance == 0.0) *result = 0.0;
    else *result = sqrt(mean - variance * variance / 8. / speed);
}

```

```

void static contB(double initial, double A, double *result)
{
    *result = sqrt(initial) - A;
}
void static contC(double A, double B, double Lambda1, double *result)
{
    if (B == 0.0 || (Lambda1 - A) == 0.0) *result = 0.0;
    else *result = -log((Lambda1 - A) / B);
}
void static funcI1(dcomplex u, double tau, double kr, dcomplex *result)
{
    *result = CRmul(CRsub(Cmul(u, CI), 1.0) , (tau + (exp(-tau * kr) - 1.) / kr) /
}
void static funcI2(dcomplex u, double tau, double kv, double sigmav, double rho1
{
    *result = Csub(RCmul(tau / (sigmav * sigmav), RCsub(kv, Cadd(RCmul(sigmav * rh
}
void static funcI3(dcomplex u, double tau, double kr, dcomplex *result)
{
    *result = RCmul(1. / (2.*pow(kr, 3.)) * (3. + exp(-2.*kr * tau) - 4.*exp(-kr *
}
void static funcI4(dcomplex u, double tau, double kr, double a, double b, double
{
    *result = RCmul(-1. / kr * (b / c * (1.0 - exp(-c * tau)) + a * tau + a / kr *
}
void static funcA(dcomplex u, double tau, double kr, double thetar, double sigma
{
    *result = Csub(Cadd(Cadd(RCmul(kr * thetar, I1), RCmul(kv * thetav, I2)), Cadd
}

void static chf(dcomplex u, double kappav, double thetav, double sigmav, double
{
    double a, b, c, fc1, fd1, flambda1, clambda;
    dcomplex expon, D1, g, I1, I2, I3, I4, A, Bx, Bv, Br;
    double tau;

    expon = Complex(0.0, 0.0);
    tau = maturity - 0.0;
    funcD1(u, kappav, sigmav, rho12, &D1);
    funcg(u, kappav, sigmav, rho12, D1, &g);
    funcC(1.0, kappav, sigmav, &fc1);

```

```

funcLambda(1.0, kappav, v0, sigmav, &flambda1);
funcD(kappav, thetav, sigmav, &fd1);
Lambda1(1.0, fc1, flambda1, fd1, &clambda);
contA(thetav, kappav, sigmav, &a);
contB(v0, a, &b);
contC(a, b, clambda, &c);
funcI1(u, tau, kappar, &I1);
funcI2(u, tau, kappav, sigmav, rho12, g, D1, &I2);
funcI3(u, tau, kappar, &I3);
funcI4(u, tau, kappar, a, b, c, &I4);
funcBx(u, &Bx);
funcBr(u, tau, kappar, sigmar, D1, g, &Br);
funcBv(u, tau, kappav, sigmav, rho12, D1, g, &Bv);
funcA(u, tau, kappar, thetar, sigmar, kappav, thetav, sigmav, rho13, dividant,
expon = Cadd(A, RCmul(v0, Bv));
expon = Cadd(expon, RCmul(r0, Br));
expon = Cadd(expon, RCmul(log(S0 / K), Bx));
*result = Cexp(expon);

}

//COSINE method to calculate inverse fourier integral
static double cosine_function_xi(double d, double c, double dma, double cma, double fnpi)
{
    double res;
    res = 1. / (1 + fnpi * fnpi) * ((cos(dma) + fnpi * sin(dma)) * exp(d) - (cos(cma) + fnpi * sin(cma)) * exp(c));
    return res;
}

static double cosine_function_psi(double d, double c, double dma, double cma, double fnpi)
{
    double res;
    res = (fnpi == 0.) ? (d - c) : (sin(dma) - sin(cma)) / fnpi;
    return res;
}

static double cosine_Vk_call(double K, double a, double b, double fnpi)
{
    double ma, bma;
    ma = -a * fnpi;
    bma = b * fnpi;

```

```

    bma = (b - a) * fnpi;
    return 2. / (b - a) * K * (cosine_function_xi(b, 0, bma, ma, fnpi) - cosine_fu
}

static double cosine_Vk_put(double K, double a, double b, double fnpi)
{
    double ma;
    ma = -a * fnpi;
    return 2. / (b - a) * K * (-cosine_function_xi(0, a, ma, 0, fnpi) + cosine_fun
}

static double cosine_Vk(int callput_flag, double K, double a, double b, double f
{
    double result;
    result = 0.0;
    if (callput_flag == 1) result = cosine_Vk_call(K, a, b, fnpi);
    else result = cosine_Vk_put(K, a, b, fnpi);
    return result;
}

void static cosine_left_bound(double L, double S0, double K, double sigmamean, d
{

    double c1, c2;
    c1 = 0.0;
    c2 = 0.0;
    c1 = r0 * tau + (1. - exp(-kappa * tau)) * (sigmamean - sigma0) / (2.0 * kappa
    c2 = (1.0 / (8.0 * pow(kappa, 3))) * (gamma1 * tau * kappa * exp(-kappa * tau)

    /*Truncation range*/
    *a = c1 - L * pow(fabs(c2), 0.5) + log(S0 / K);
    *b = c1 + L * pow(fabs(c2), 0.5) + log(S0 / K);
}

/*Compute Price Option*/
int AP_FourierCosine_HesVas(int callput_flag, double S0, NumFunc_1 *p, double
{
    double sum, sumd, a, b, vk, x, invbma, xma;
    dcomplex fnpi, phi;
    double rho12, rho13;
    double L;

```

```

int i;
double K;

K = p->Par[0].Val.V_PDOUBLE;
L = 10; //Integration Domain
rho12 = rhoSv;
rho13 = rhoSr;
sum = 0;
sumd = 0;
a = 0;
b = 0;
x = log(S0 / K);
invbma = 0.0;
xma = 0.0;
fnpi = CZERO;
phi = CZERO;

cosine_left_bound(L, S0, K, thetav, v0, sigmar, sigmav, kappav, r0, maturity,

xma = x - a;
invbma = M_PI / (b - a);
chf(fnpi, kappav, thetav, sigmav, v0, kappar, thetar, sigmar, r0, rho12, rho13
vk = cosine_Vk(callput_flag, K, a, b, fnpi.r);
sum = 0.5 * phi.r * vk;
sumd = 0.;

for (i = 1; i < N; i++)
{
    fnpi.r += invbma;
    chf(fnpi, kappav, thetav, sigmav, v0, kappar, thetar, sigmar, r0, rho12, r
    vk = cosine_Vk(callput_flag, K, a, b, fnpi.r);
    sum += (phi.r * cos(fnpi.r * xma) - phi.i * sin(fnpi.r * xma)) * cosine_Vk
    sumd -= (phi.i * cos(fnpi.r * xma) + phi.r * sin(fnpi.r * xma)) * fnpi.r *
}
/*GreekDelta=sumd/S0;
*ptprice = sum;

return OK;
}

int CALC(AP_FOURIERCOSINE_HESVAS)(void *Opt, void *Mod, PricingMethod *Met)

```

```

{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double divid;
    int iscall;

    divid = ptMod->divid.Val.V_DOUBLE;
    iscall = FALSE;
    if (ptOpt->PayOff.Val.V_NUMFUNC_1->Compute == &Call) iscall = TRUE;

    return AP_FourierCosine_HesVas(iscall, ptMod->S0.Val.V_PDOUBLE,
                                   ptOpt->PayOff.Val.V_NUMFUNC_1,
                                   ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                                   ptMod->r0.Val.V_PDOUBLE,
                                   , ptMod->kr.Val.V_PDOUBLE,
                                   ptMod->thetar.Val.V_PDOUBLE,
                                   ptMod->Sigmar.Val.V_PDOUBLE,
                                   ptMod->V0.Val.V_PDOUBLE,
                                   , ptMod->kV.Val.V_PDOUBLE,
                                   ptMod->thetaV.Val.V_PDOUBLE,
                                   ptMod->SigmaV.Val.V_PDOUBLE,
                                   ptMod->RhoSr.Val.V_PDOUBLE,
                                   ptMod->RhoSV.Val.V_PDOUBLE,
                                   ptMod->RhorV.Val.V_PDOUBLE,
                                   Met->Par[0].Val.V_PINT,
                                   &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(AP_FOURIERCOSINE_HESVAS)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt)->Name, "PutEuro") == 0))
        return OK;
    return WRONG;
}
#endif //PremiaCurrentVersion

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

```

```

        Met->HelpFilenameHint = "ap_fouriercosine_hesvas";
        Met->Par[0].Val.V_INT2 = 128;
    }

    return OK;
}

PricingMethod MET(AP_FOURIERCOSINE_HESVAS) =
{
    "AP_FOURIERCOSINE_HESVAS",
    {"Number of integration steps (power of 2)", INT2, {100}, ALLOW}, {" ", PREMI
    CALC(AP_FOURIERCOSINE_HESVAS),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_FOURIERCOSINE_HESVAS),
    CHK_ok,
    MET(Init)
};

```