

[Help](#)

```
#include "
href../../../../mod/heshw2d/heshw2d_std/heshw2d_std_h_src.pdfheshw2d_std.h"
#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"
#include "pnl/pnl_integration.h"

//The "#else" part of the code will be freely available after the (year of creat
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2015+2) //The "#els
static int CHK_OPT(AP_FOURIERCOSINE_HESHW2D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_FOURIERCOSINE_HESHW2D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

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

/*ZCB Data*/
static double *tm; /*Times T of maturities read in the file initialyield.dat */
static double *Pm; /*Values of the zero coupon P(0,tm) read in the file initialy

static int lecture_tr(int flag_rd)
{
    int i;
    char ligne[30];
    char *pligne;
    double p, tt_value;
```

```

FILE *Entrees;

if (flag_rd)
    Entrees = fopen(init_tr_rd, "r");
else
    Entrees = fopen(init_tr_rf, "r");

    if (Entrees == NULL)
    {
        printf("LE FICHIER N'A PU ETRE OUVERT. VERIFIEZ LE CHEMIN.\ n");
    }

    /* i is the number of lines that has been read */
    i = 0;
    pligne = ligne;
    Pm = (double *)malloc(200 * sizeof(double));
    tm = (double *)malloc(200 * sizeof(double));

    while (1)
    {
        pligne = fgets(ligne, sizeof(ligne), Entrees);
        if (pligne == NULL) break;
        else
        {
            sscanf(ligne, "%lf t=%lf", &p, &tt_value);
            /* The line read must be written "0.943290 t=0.5" where 0.943290 is
            Pm[i] = p; /*save the price of the zero coupon*/
            tm[i] = tt_value; /*save the corresponding time*/
            i++;

        }
    }
    fclose(Entrees);
    return i;
}

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), k
}
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, RCmul(gamma*rho12*u, CI)), fd),
}
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, RCsub(1
}
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+fLambda)));
}
void static funcbeta1(double kappa, double vbar, double gamma, double *result)
{
if(vbar-gamma*gamma/8./kappa<0.0)      { *result =0.0; printf("Invalid paramete
    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(-beta30*x))*(
}
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(-lambdad0*(T0-x)
}
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(-lambdaf0*(T0-x)
}
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(-beta30*x))
}
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(-lambdad
}

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(-lambdad0*
    }
static double intgfzeta(double x, void *p)
{
    return (rho130*etad0*(exp(-lambdad0*(T0-x))-1.)/lambdad0-rho140*etaf0*(e
}

void static funcA(double u, double v0, double kappa, double vbar, double gammav,
{
double intg1r, intg2r, intg3r, intg1i, intg2i, intg3i, intgzeta;
    double fC1, fLambda, fD1, Lambda1, beta1, beta2, beta3;
    dcomplex fd, fg;
PnlFunc func1r, func2r, func3r, func1i, func2i, func3i, funczeta;
int n;

    n=50;
    intg1r=0.;
    intg2r=0.;
    intg3r=0.;
    intg1i=0.;
    intg2i=0.;
    intg3i=0.;
    intgzeta=0.;

    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=intgfheta;
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*intg2r+u*intg3r, RCmul(intgzeta, u))), RCmul(intgzeta, u));
}

void static chf(double u, double v0, double kappa, double vbar, double gammav, double 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, etad, etaf,
expon = Cadd( Cadd(fa, RCmul(v0, fc)) , RCmul(log(csi0*(pf0/pd0)/strike), fb)

```

```

*result= Cexp(expon);
}

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

void static cosine_bound(double L, double csi0, double strike, double pd0, doubl
{
    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);

```



```

double pd0, pf0;
int    NZCB, Ndivident, i;
int    flag_rd;

strike = p->Par[0].Val.V_PDDOUBLE;

//parameters of the model
kappav = GET(kappa, 0);
kappar = GET(kappa, 1);
kappaq = GET(kappa, 2);

sigmav = GET(sigma, 0);
sigmar = GET(sigma, 1);
sigmaq = GET(sigma, 2);

rhoSv = GET(rho, 0);
rhoSr = GET(rho, 1);
rhoSq = GET(rho, 2);
rhovr = GET(rho, 3);
rhovq = GET(rho, 4);
rhorq = GET(rho, 5);

if ((fabs(rhovr) > 0) || (fabs(rhovq) > 0) || (fabs(rhorq) > 0))
    return UNTREATED_CASE;

init_tr_rd = curve_rd;
init_tr_rf = curve_rf;

if((flat_flag_rd==0)&&(flat_flag_rf==0))
{
pd0=exp(-r0*T); // zero coupond bond price for flat case
pf0=exp(-q0*T); // zero coupond bond price of dividend for flat dividend
}
else
{
if(flat_flag_rd==0)
    pd0=exp(-r0*T);
else
{
    flag_rd = 1;
    NZCB=lecture_tr(flag_rd);

```

```

    if (T > tm[NZCB - 1])
    {
        printf("\ nError : Maturity bigger than the last time value entered in 'initia
        pd0=1.;
    }
    else
    {
        for (i=0; i<NZCB-1; i++)
        {
            if(tm[i]==T)
            {
                pd0=Pm[i];
            }
            else if (tm[i]< T && tm[i+1]> T)
            {
                pd0=Pm[i]+ (T-tm[i])/(tm[i+1]-tm[i])*(Pm[i+1]-Pm[i]);
            }
        }
    }
}

if(flat_flag_rf==0)
    pf0=exp(-q0*T);
    else
    {
        flag_rd=0;
        Ndivident=lecture_tr(flag_rd);
        if (T > tm[Ndivident - 1])
        {
            printf("\ nError : Maturity bigger than the last time value entered in 'divide
            pf0=1.;
        }
        else
        {
            for (i=0; i<Ndivident-1; i++)
            {
                if(tm[i]==T)
                {
                    pf0=Pm[i];
                }
            }
            else if (tm[i]< T && tm[i+1]> T)

```

```

{
    pf0=Pm[i]+ (T-tm[i])/(tm[i+1]-tm[i])*(Pm[i+1]-Pm[i]);
}
}
}
}
    free(Pm);
    free(tm);
}

```

HHW4d(v0, kappav, thetav, sigmav, pd0, kappar, sigmar, pf0, kappaq, sigmaq, rho

```

//Price
*ptprice =Price;

return OK;
}

```

```

int CALC(AP_FOURIERCOSINE_HESHW2D)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    int iscall;
    iscall = FALSE;
    if (ptOpt->PayOff.Val.V_NUMFUNC_1->Compute == &Call) iscall = TRUE;
    return AP_FourierCosin_HesHw2d(iscall,ptMod->S0.Val.V_PDOUBLE,
                                   ptOpt->PayOff.Val.V_NUMFUNC_1,
                                   ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                                   ptMod->v0.Val.V_PDOUBLE,
                                   ptMod->flat_flag_rd.Val.V_INT,
                                   MOD(GetYield_rd)(ptMod),
                                   MOD(GetCurve_rd)(ptMod),
                                   ptMod->flat_flag_rf.Val.V_INT,
                                   MOD(GetYield_rf)(ptMod),
                                   MOD(GetCurve_rf)(ptMod),
                                   ptMod->theta.Val.V_PDOUBLE,
                                   ptMod->kappa.Val.V_PNLVECT,
                                   ptMod->sigma.Val.V_PNLVECT,
                                   ptMod->rho.Val.V_PNLVECT,
                                   Met->Par[0].Val.V_PINT,

```

```

&(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(AP_FOURIERCOSINE_HESHW2D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt)-
        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_heshw2d";
        Met->Par[0].Val.V_INT = 128;
    }

    return OK;
}

PricingMethod MET(AP_FOURIERCOSINE_HESHW2D) =
{
    "AP_FourierCosine",
    { {"Number of integration steps (power of 2)", INT2, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(AP_FOURIERCOSINE_HESHW2D),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_FOURIERCOSINE_HESHW2D),
    CHK_ok,
    MET(Init)
};

```