

[Help](#)

```
#include "
href../../../../mod/hhw4d/hhw4d_stdh/hhw4d_stdh_h_src.pdfhhw4d_stdh.h"
#include <
href../../../../common/math/cdo/cdo_math_h_src.pdfmath.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_INF)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_HHW4D_INF)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

struct tmp_parameters
{
    double u_p, kappa_p, vbar_p, rho12_p, rho13_p, rho14_p, rho34_p, sigmav_p, rho
    dcomplex fd_p, fg_p;
};
static void FuncBn(double an, double s, double t, double *result)
{
    *result = (1.0 - exp(-an * (t - s))) / an;
}
static void FuncBr(double ar, double s, double t, double *result)
{
    *result = (1.0 - exp(-ar * (t - s))) / ar;
}
static void 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
}
static void 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), C
}
static void FuncC(double u, dcomplex fd, dcomplex fg, double tau, double kappa,
{
    *result = Cmul(Cdiv(RCsub(1., Cexp(RCmul(-tau, fd))), RCmul(gamma * gamma, RCs
}
static void 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, sigmav, kappa!\n n vbar < sigmav*sigmav
    }
    else if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
    else *result = sqrt(vbar - gamma * gamma / 8. / kappa);
}
static void funcbeta2(double v0, double beta1, double *result)
{
    *result = sqrt(v0) - beta1;
}
static void funcbeta3(double kappa, double gamma, double vbar, double v0, double
{
    double fC1, fD1, fLambda, Lambda1;
    fC1 = gamma * gamma * (1. - exp(-kappa)) / (4.*kappa);
    fD1 = 4.*kappa * vbar / (gamma * gamma);
    fLambda = 4.*kappa * v0 * exp(-kappa) / (gamma * gamma * (1. - exp(-kappa)));
    Lambda1 = sqrt(fC1 * (fLambda - 1.) + fC1 * fD1 + fC1 * fD1 / (2.*(fD1 + fLamb
    if (beta2 == 0.0 || (Lambda1 - beta1) == 0.0) *result = 0.0;
    else *result = -log((Lambda1 - beta1) / beta2);
}

static void FLambda(double rho14, double etar, double an, double ar, double s, d
{
    *result = (beta1 + beta2 * exp(-beta3 * t)) * rho14 * etar / ar * (t - s - Br
}
static void FuncAr(double rr0, double ar, double etar, double s, double t, doubl

```

```

{
    double pr1, pr2;
    pr1 = exp(- rr0 * (s));
    pr2 = exp(- rr0 * (t));
    *result = log(pr2 / pr1) * (Br * rr0 - etar * etar / 4. / ar * (1.0 - exp(-2.*
}
static void FuncAn(double rn0, double an, double etan, double s, double t, doubl
{
    double pr1, pr2;
    pr1 = exp(- rn0 * (s));
    pr2 = exp(- rn0 * (t));
    *result = log(pr2 / pr1) * (Bn * rn0 - etan * etan / 4. / an * (1.0 - exp(-2.*
}

static double intgf1r(double x, void *p)
{
    struct tmp_parameters *data ;
    dcomplex fC;

    data = (struct tmp_parameters *) p;
    FuncC(data->u_p, data->fd_p, data->fg_p, x, data->kappa_p, data->sigmav_p, dat
    return (data->kappa_p * data->vbar_p + data->rho23_p * data->sigmav_p * data->
}
static double intgf2r(double x, void *p)
{
    struct tmp_parameters *data ;
    dcomplex fC;

    data = (struct tmp_parameters *) p;
    FuncC(data->u_p, data->fd_p, data->fg_p, x, data->kappa_p, data->sigmav_p, dat
    return data->rho23_p * data->etan_p * data->sigmav_p * (data->beta1_p + data->
}
static double intgf3r(double x, void *p)
{
    struct tmp_parameters *data ;
    dcomplex fC;

    data = (struct tmp_parameters *) p;
    FuncC(data->u_p, data->fd_p, data->fg_p, x, data->kappa_p, data->sigmav_p, dat
    return data->rho24_p * data->sigmav_p * data->etar_p * (data->beta1_p + data->
}

```

```

static double intgf1i(double x, void *p)
{
    struct tmp_parameters *data ;
    dcomplex fC;

    data = (struct tmp_parameters *) p;
    FuncC(data->u_p, data->fd_p, data->fg_p, x, data->kappa_p, data->sigmav_p, data->rho23_p, data->etan_p, data->beta1_p, data->rho13_p, data->rho14_p);
    return (data->kappa_p * data->vbar_p + data->rho23_p * data->sigmav_p * data->etan_p + data->rho13_p * data->beta1_p + data->rho14_p);
}

static double intgf2i(double x, void *p)
{
    struct tmp_parameters *data ;
    dcomplex fC;

    data = (struct tmp_parameters *) p;
    FuncC(data->u_p, data->fd_p, data->fg_p, x, data->kappa_p, data->sigmav_p, data->rho23_p, data->etan_p, data->beta1_p, data->rho13_p, data->rho14_p);
    return data->rho23_p * data->etan_p * data->sigmav_p * (data->beta1_p + data->rho13_p + data->rho14_p);
}

static double intgf3i(double x, void *p)
{
    struct tmp_parameters *data ;
    dcomplex fC;

    data = (struct tmp_parameters *) p;
    FuncC(data->u_p, data->fd_p, data->fg_p, x, data->kappa_p, data->sigmav_p, data->rho23_p, data->etan_p, data->beta1_p, data->rho13_p, data->rho14_p);
    return data->rho24_p * data->sigmav_p * data->etar_p * (data->beta1_p + data->rho13_p + data->rho14_p);
}

static double intgfzeta(double x, void *p)
{
    struct tmp_parameters *data ;
    data = (struct tmp_parameters *) p;
    return (data->rho13_p * data->etan_p * (exp(-data->an_p * (data->T_p - x)) - 1));
}

void read_para(double u, double kappa, double vbar, double rho12, double rho13, double rho14)
{
    (* partptr).u_p = u;
    (* partptr).kappa_p = kappa;
    (* partptr).vbar_p = vbar;
    (* partptr).rho12_p = rho12;
    (* partptr).rho13_p = rho13;
    (* partptr).rho14_p = rho14;
}

```

```

(* partptr).rho34_p = rho34;
(* partptr).sigmav_p = sigmav;
(* partptr).rho23_p = rho23;
(* partptr).etan_p = etan;
(* partptr).an_p = an;
(* partptr).rho24_p = rho24;
(* partptr).etar_p = etar;
(* partptr).ar_p = ar;
(* partptr).beta1_p = beta1;
(* partptr).beta2_p = beta2;
(* partptr).beta3_p = beta3;
(* partptr).T_p = t - s;
(* partptr).fd_p = fd;
(* partptr).fg_p = fg;
}

static void FuncA(double u, double v0, double kappa, double vbar, double sigmav,
{
    double intg1r = 0., intg2r = 0., intg3r = 0., intg1i = 0., intg2i = 0., intg3i
    double beta1, beta2, beta3;
    dcomplex fd, fg;
    PnlFunc func1r, func2r, func3r, func1i, func2i, func3i, funczeta;
    int n = 50;
    struct tmp_parameters parameter;

    Funcd(u, kappa, sigmav, rho12, &fd);
    Funcg(u, kappa, sigmav, rho12, fd, &fg);
    funcbeta1(kappa, vbar, sigmav, &beta1);
    funcbeta2(v0, beta1, &beta2);
    funcbeta3(kappa, sigmav, vbar, v0, beta1, beta2, &beta3);
    read_para(u, kappa, vbar, rho12, rho13, rho14, rho34, sigmav, rho23, etan, an,

    func1r.F = intgf1r;
    func2r.F = intgf2r;
    func3r.F = intgf3r;
    func1i.F = intgf1i;
    func2i.F = intgf2i;
    func3i.F = intgf3i;
    funczeta.F = intgfzeta;
    func1r.params = &parameter;
    func2r.params = &parameter;
    func3r.params = &parameter;

```

```

func1i.params = &parameter;
func2i.params = &parameter;
func3i.params = &parameter;
funczeta.params = &parameter;
intg1r = pnl_integration(&func1r, 0.0, t - s, n, "simpson");
intg2r = pnl_integration(&func2r, 0.0, t - s, n, "simpson");
intg3r = pnl_integration(&func3r, 0.0, t - s, n, "simpson");
intg1i = pnl_integration(&func1i, 0.0, t - s, n, "simpson");
intg2i = pnl_integration(&func2i, 0.0, t - s, n, "simpson");
intg3i = pnl_integration(&func3i, 0.0, t - s, n, "simpson");
intgzeta = pnl_integration(&funczeta, 0.0, t - s, n, "simpson");

*result = Cadd(RCadd(intg1r + u * intg2i - u * intg3i, RCmul(intgli - u * intg
}

static void FuncAlpha(double t, double rl0, double etal, double al, double *result)
{
    *result = rl0 + etal * etal / 2. / al / al * (1. - exp(-al * t)) * (1. - exp(-
}
static void Mean_rn(double rns, double an, double etan, double alphans, double a
{
    *result = rns * exp(-an * (t - s)) + alphant - alphans * exp(-an * (t - s)) +
}
static void Mean_rr(double rrs, double ar, double an, double alphars, double alp
{
    *result = rrs * exp(-ar * (t - s)) + alphart - alphars * exp(-ar * (t - s)) +
}
static void Var_rn(double etan, double an, double s, double t, double *result)
{
    *result = etan * etan / (2.*an) * (1.0 - exp(-2.*an * (t - s)));
}
static void Var_rr(double etar, double ar, double s, double t, double *result)
{
    *result = etar * etar / (2.*ar) * (1.0 - exp(-2.*ar * (t - s)));
}
static void Cov(double rhonr, double var_rn, double var_rr, double *result)
{
    *result = rhonr * sqrt(var_rn * var_rr);
}
static void Func_expec1(double u, double kappa, double sigmav, double vbar, doub

```

```

{
    double alpha_ns, alpha_nt, alpha_rs, alpha_rt, mrn, mrr, vrn, vrr, corr, mean,
    FuncAlpha(t0, rn0, etan, an, &alpha_ns);
    FuncAlpha(s, rn0, etan, an, &alpha_nt);
    FuncAlpha(t0, rr0, etar, ar, &alpha_rs);
    FuncAlpha(s, rr0, etar, ar, &alpha_rt);
    Mean_rn(rn0, an, etan, alpha_ns, alpha_nt, t0, s, t, &mrn);
    Mean_rr(rr0, ar, an, alpha_rs, alpha_rt, rho34, rho14, beta1, beta2, beta3, et
    Var_rn(etan, an, t0, s, &vrn);
    Var_rr(etar, ar, t0, s, &vrr);
    Cov(rho34, vrn, vrr, &corr);
    mean = Bn * mrn - Br * mrr;
    variance = Bn * Bn * vrn + Br * Br * vrr - 2.*Bn * Br * corr;
    *result = Cexp(CRsub(RCmul(u * mean, CI), -0.5 * u * u * variance));
}

static void Func_psi(double u, double kappa, double sigmav, double vbar, double
{
    *result = RCdiv(1., RCsub(1., RCmul(2.*sigmav * sigmav / 4. / kappa * (1. - ex
}
static void Func_expec2(double u, double kappa, double sigmav, double vbar, doub
{
    *result = Cmul(Cpow_real(fpsi, 2.*kappa * vbar / sigmav / sigmav), RCmul(exp(-
}

static void chf(double u, double v0, double kappa, double vbar, double sigmav, d
{
    double Bn, Br, beta1, beta2, beta3, Lambda, fAn, fAr;
    dcomplex fd, fg, fpsi, fA, fC, expec1, expec2;

    FuncBn(an, s, t, &Bn);
    FuncBr(ar, s, t, &Br);
    funcbeta1(kappa, vbar, sigmav, &beta1);
    funcbeta2(v0, beta1, &beta2);
    funcbeta3(kappa, sigmav, vbar, v0, beta1, beta2, &beta3);
    FLambda(rho14, etar, an, ar, s, t, beta1, beta2, beta3, Bn, Br, &Lambda);
    FuncAr(rr0, ar, etar, s, t, Br, Lambda, &fAr);
    FuncAn(rn0, an, etan, s, t, Bn, &fAn);
    FuncA(u, v0, kappa, vbar, sigmav, rho12, rho13, rho14, rho23, rho24, rho34, et
    Func_expec1(u, kappa, sigmav, vbar, v0, rn0, an, etan, rr0, ar, etar, rho14, r
    Funcd(u, kappa, sigmav, rho12, &fd);

```

```

Funcg(u, kappa, sigmav, rho12, fd, &fg);
FuncC(u, fd, fg, t - s, kappa, sigmav, rho12, &fC);
Func_psi(u, kappa, sigmav, vbar, s, t, fC, &fpsi);
Func_expec2(u, kappa, sigmav, vbar, s, t, v0, fpsi, fC, &expec2);
*result = Cmul(Cmul(Cexp(Cadd(RCmul(u * (fAr - fAn) - u * log(strike), CI), fA
}

//COSINE method to calculate inverse fourier integral
static double cosine_function_xi(double d, double c, double dma, double cma, dou
{
    double res = 1. / (1 + fnpi * fnpi) * ((cos(dma) + fnpi * sin(dma)) * exp(d) -
    return res;
}
static double cosine_function_psi(double d, double c, double dma, double cma, do
{
    double res = (fnpi == 0.) ? (d - c) : (sin(dma) - sin(cma)) / fnpi;
    return res;
}
static 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;
}

static void inflation_HHW(double v0, double kappa, double vbar, double sigmav, d
{
    double cap, sum, K, a, b;
    double invbma, fnpi, tk1, tk2;
    dcomplex phi;
    int N, M, i, j;

    a = -10.;
    b = 10;
    N = pow(2, 10);
    K = strike + 1;
    M = (T - t0) / tau;
    cap = 0.;
    for (j = 0; j < M; j++)
    {

```



```

    tk1 = t0 + j * tau;
    tk2 = tk1 + tau;
    invbma = 0.0;
    fnpi = 0.0;
    invbma = M_PI / (b - a);
    chf(fnpi, v0, kappa, vbar, sigmav, rn0, etan, an, rr0, etar, ar, rho12, rho13, rho14, rho23, rho24, rho34, rho43, rho44);
    sum = 0.5 * phi.r * cosine_Vk(K, a, b, fnpi);
    for (i = 1; i < N; i++)
    {
        fnpi += invbma;
        chf(fnpi, v0, kappa, vbar, sigmav, rn0, etan, an, rr0, etar, ar, rho12, rho13, rho14, rho23, rho24, rho34, rho43, rho44);
        sum += (phi.r * cos(fnpi * (-a)) - phi.i * sin(fnpi * (-a))) * cosine_Vk(K, a, b, fnpi);
    }
    cap += sum * exp(-an * tk2);
}
*Price = cap;
}

```

```

static int ap_hhw4d_inf(NumFunc_1 *p, double cap_maturity, double strike, double
{
    double v0;
    double sigmav, etan, etar;
    double kappa, an, ar;
    double rho12, rho13, rho14, rho23, rho24, rho34;
    double rn0, rr0;
    double t0, tau, T, Price;

    t0 = first_reset_date;
    tau = tenor;
    T = cap_maturity;

    //-----Initialisation of variable
    v0 = GET(x0, 1);

    rn0 = GET(r, 0);
    rr0 = GET(r, 1);

    kappa = GET(kappa_v, 0);
    an = GET(kappa_v, 1);
}

```

```

ar = GET(kappa_v, 2);

sigmav = GET(sigma, 0);
etan = GET(sigma, 1);
etar = 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);

inflation_HHW(v0, kappa, vbar, sigmav, rn0, an, etan, rr0, ar, etar, rho12, rho13, rho14, rho23, rho24, rho34)

*ptprice = Price;
return OK;
}

int CALC(AP_HHW4D_INF)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return ap_hhw4d_inf(ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->BMaturity.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_INF)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "InflationIndexedCap") == 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_inflation";
    }

    return OK;
}

PricingMethod MET(AP_HHW4D_INF) =
{
    "AP_HHW4D_INF",
    {{ " ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_HHW4D_INF),
    { {"Price", DOUBLE, {100}, FORBID},
      { " ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_HHW4D_INF),
    CHK_ok,
    MET(Init)
};

```