

[Help](#)

```

#include <stdlib.h>
#include "cirpp1d_std.h"
#include "pnl/pnl_vector.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(TR_SwaptionCIRpp1D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_SwaptionCIRpp1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/*////////////////////////////////////// Datas specific to Hull and White //

static double a;                /*Speed reversion of the Hullwhite model
static double b;
static double rx0;
static double sigma;            /*Volatility of the Hullwhite model.*/

static struct Tree Tr;           /* The unique tree variable create by Pr

static double VarTree(double s)
{
    return s * s;
}

static double Var_y(double s)
{
    /*Variation of the variable tree y at time s (must be independent of a variabl
    double V;

    V = sigma * sqrt(s) / 2.0;
    return V;
}

```

```

static double ExpectCond_y(double x0, double s)
{
    /*Conditional expectation of variable y used in tree at time s starting from t
    double E, x00;
    x00 = 0.5 * sqrt(s * (4 * a * b - sigma * sigma) / (2 - a * s));

    E = x0 + ((a * b / 2 - sigma * sigma / 8) / x0 - a * x0 / 2.0) * s;
    if (x0 < x00)
    {
        E = x00 + ((a * b / 2 - sigma * sigma / 8) / x00 - a * x00 / 2.0) * s;
    }

    return E;
}

```

```

static int indiceTime(struct Tree *Meth, double s)
{
    int i = 0;

    if (Meth->t == NULL)
    {
        printf("FATALE ERREUR, PAS DE GRILLE DE TEMPS !");
    }
    else
    {
        while (Meth->t[i] <= s && i <= Meth->Ngrid)
        {
            i++;
        }
    }
    return i - 1;
}

```

```

static int SetTimegrid(struct Tree *Meth, int n, double T)
{
    int i;

    Meth->Ngrid = n;
}

```

```

Meth->Tf = T;

Meth->t = malloc((Meth->Ngrid + 1) * sizeof(double));

for (i = 0; i < Meth->Ngrid + 1; i++)
{
    Meth->t[i] = i * Meth->Tf / Meth->Ngrid;
}

return 1;
}

static int DeleteTimegrid(struct Tree *Meth)
{
    free(Meth->t);
    return 1;
}

static void SetTree(struct Tree *Meth)
{
    int jmin, jmax, jminprev, jmaxprev;
    double x, xi;
    int h, i, j, k, nv;

    double M, sigma_i, mu_jk, M_ij, dx;
    if (Meth->t == NULL)
    {
        printf("FATAL ERROR IN SetTree(), SetTimegrid must be used before SetTree\n");
    }

    jmin = 0;
    jmax = 0;
    xi = 0;
    nv = 1;
    /* Allocation of all the tree variable*/
    Meth->pLRij = malloc((Meth->Ngrid + 1) * sizeof(double *));
    Meth->pLPDo = malloc((Meth->Ngrid) * sizeof(double *));
    Meth->pLPMi = malloc((Meth->Ngrid) * sizeof(double *));
    Meth->pLPUp = malloc((Meth->Ngrid) * sizeof(double *));

```

```

Meth->pLRef = malloc((Meth->Ngrid) * sizeof(int *));
Meth->>TSize = malloc((Meth->Ngrid + 1) * sizeof(int));

Meth->pLRij[0] = malloc(sizeof(double));

Meth->pLRij[0][0] = xi;
Meth->>TSize[0] = 1;

/* one step backward translation of the tree, there are 3 point in rank 0 for
{
    jmin = -1;
    jmax = +1;
    xi = 0;
    nv = 3;
    free(Meth->pLRij[0]);
    Meth->pLRij[0] = malloc(3 * sizeof(double));
    Meth->pLRij[0][0] = -sqrt(3.) * Var_y(Meth->t[1]);
    Meth->pLRij[0][1] = xi;
    Meth->pLRij[0][2] = +sqrt(3.) * Var_y(Meth->t[1]);
    Meth->>TSize[0] = 3;
}

/* iteration on the time step */
for (i = 1; i <= Meth->Ngrid; i++)
{
    sigmai = Var_y(Meth->t[i] - Meth->t[i - 1]);
    dx = sqrt(3.) * sigmai;
    xi = ExpectCond_y(xi, Meth->t[i] - Meth->t[i - 1]);
    jminprev = jmin;
    jmaxprev = jmax;

    M = ExpectCond_y(Meth->pLRij[i - 1][0], Meth->t[i] - Meth->t[i - 1]);
    jmin = pnl_iround((M - xi) / dx) - 1;
    M = ExpectCond_y(Meth->pLRij[i - 1][nv - 1], Meth->t[i] - Meth->t[i - 1]);
    jmax = pnl_iround((M - xi) / dx) + 1;

    Meth->pLPDo[i - 1] = malloc(nv * sizeof(double));
    Meth->pLPMi[i - 1] = malloc(nv * sizeof(double));
    Meth->pLPUi[i - 1] = malloc(nv * sizeof(double));
    Meth->pLRef[i - 1] = malloc(nv * sizeof(int));

```

```

nv = jmax - jmin + 1;
Meth->TSize[i] = nv;

Meth->pLRij[i] = malloc(nv * sizeof(double));

for (k = jmin; k <= jmax; k++)
{
    j = k - jmin;

    x = k * dx + xi;
    Meth->pLRij[i][j] = x;

}
for (k = jminprev; k <= jmaxprev; k++)
{
    j = k - jminprev;
    Mij = ExpectCond_y(Meth->pLRij[i - 1][j], Meth->t[i] - Meth->t[i - 1])
    h = pnl_iround((Mij - xi) / dx);

    mujk = Mij - h * dx - xi;

    Meth->pLPU[i - 1][j] = 1. / 6. + pow(mujk / dx, 2) / 2. + mujk / (2.*
    Meth->pLPMi[i - 1][j] = 2. / 3. - pow(mujk / dx, 2);
    Meth->pLPDo[i - 1][j] = 1. / 6. + pow(mujk / dx, 2) / 2. - mujk / (2.*
    Meth->pLRef[i - 1][j] = h - jmin;

    if (h <= jmin)
    {
        printf("ERROR FATAL JMIN JMAX IN SetTree(), ExpectCond_y() MUST BE
    }
    if (h >= jmax)
    {
        printf("ERROR FATAL JMIN JMAX IN SetTree(), ExpectCond_y() MUST BE
    }

}

}

```

```

    /*printf("FIN de la construction de l'arbre des taux\ n");*/
}

static void TranslateTree(struct Tree *Meth, ZCMarketData *ZCMarket)
{
    int k, i, j;
    double alpha, sum, eps;

    if (Meth->t == NULL)
    {
        printf("FATAL ERROR IN TranslateTree(), SetTimegrid() and SetTree() must b
    }
    if (Meth->pLRij == NULL)
    {
        printf("FATAL ERROR IN TranslateTree(), SetTimegrid() and SetTree() must b
    }

    eps = Meth->Tf / Meth->Ngrid;
    alpha = -log(BondPrice(eps, ZCMarket)) / eps;

    Meth->pLQij = malloc((Meth->Ngrid + 1) * sizeof(double *));
    Meth->pLQij[0] = malloc(sizeof(double));
    Meth->pLQij[0][0] = 1.;

    {
        free(Meth->pLQij[0]);
        Meth->pLQij[0] = malloc(3 * sizeof(double));
        Meth->pLQij[0][0] = 0;
        Meth->pLQij[0][1] = 1.;
        Meth->pLQij[0][2] = 0;
    }

    /* Recalculate the 'x' the translated short rate variable in the tree : x=Var
    for (i = 0; i < Meth->Ngrid + 1; i++)
    {
        for (j = 0; j < Meth->TSize[i]; j++)
        {
            Meth->pLRij[i][j] = VarTree(Meth->pLRij[i][j]);

```

```

    }
}

/* Iteration for alpha translation to obtain the real short rate variable r in
for (i = 0; i < Meth->Ngrid; i++)
{
    Meth->P_T = 0.0;
    Meth->pLQij[i + 1] = malloc(Meth->TSize[i + 1] * sizeof(double));

    for (j = 0; j < Meth->TSize[i]; j++)
    {
        Meth->pLRij[i][j] += alpha;
    }

    for (j = 0; j < Meth->TSize[i + 1]; j++)
    {
        sum = 0.0;
        for (k = 0; k < Meth->TSize[i]; k++)
        {
            if (Meth->pLRef[i][k] == j - 1)
            {
                sum += (Meth->pLPUp[i][k] * Meth->pLQij[i][k] * exp(-Meth->pL
            }
            if (Meth->pLRef[i][k] == j)
            {
                sum += (Meth->pLPMi[i][k] * Meth->pLQij[i][k] * exp(-Meth->pL
            }
            if (Meth->pLRef[i][k] == j + 1)
            {
                sum += (Meth->pLPDo[i][k] * Meth->pLQij[i][k] * exp(-Meth->pL
            }
        }

        Meth->pLQij[i + 1][j] = sum;
        Meth->P_T = Meth->P_T + sum;
    }
}

```

```

        sum = 0;
        for (j = 0; j < Meth->TSize[i + 1]; j++)
        {
            sum += Meth->pLQij[i + 1][j] * exp(-(Meth->t[i + 1] - Meth->t[i]) * Me

        }

        sum = sum / BondPrice(Meth->t[i + 1] + eps, ZCMarket);
        alpha = log(sum) / (Meth->t[i + 1] - Meth->t[i]);

    }

    /* Last time step alpha translation */
    for (j = 0; j < Meth->TSize[Meth->Ngrid]; j++)
    {
        Meth->pLRij[Meth->Ngrid][j] = VarTree(Meth->pLRij[Meth->Ngrid][j]);
        Meth->pLRij[Meth->Ngrid][j] += alpha;
    }

    /*printf("FIN de la translation de l'arbre des taux, sum = %f\ n", Meth->P_T);
}

static void Computepayoff(struct Tree *Meth, double s)
{
    double ht;
    int i, j, i_end;
    i_end = indiceTime(Meth, s);

    if (Meth->t == NULL)
    {
        printf("FATAL ERROR IN Computepayoff(), SetTimegrid() and SetTree() must b
    }
    if (Meth->pLRij == NULL)
    {
        printf("FATAL ERROR IN Computepayoff(), SetTimegrid() and SetTree() must b
    }

    if (Meth->Payofffunc == NULL)
    {
        initPayoff1_tr(Meth, Meth->Tf);
    }

```



```

    printf("DEFAULT PAYOFF 1\ n"); /*Payoff 1 par default.*/
}

/* pLQij[i_end][j] register the payoff at expiry time */
for (j = 0; j < Meth->TSize[i_end]; j++)
{
    Meth->pLQij[i_end][j] = Meth->Payoffunc[i_end][j];
}
/* Computation in pLQij[i][j] of the value of payoff at time step i, backward
for (i = i_end - 1; i >= 0; i--)
{
    for (j = 0; j < Meth->TSize[i]; j++)
    {

        ht = 0;
        ht = exp(- Meth->pLRij[i][j] * (Meth->t[i + 1] - Meth->t[i]));
        ht = ht * (Meth->pLPDo[i][j] * (Meth->pLQij[i + 1][ Meth->pLRef[i][j]
            + Meth->pLPMi[i][j] * (Meth->pLQij[i + 1][ Meth->pLRef[i][j]
            + Meth->pLPUi[i][j] * (Meth->pLQij[i + 1][ Meth->pLRef[i][j]

        /* Compare, in case of american, the computed value with the under nex
        if (ht < Meth->Payoffunc[i][j])
        {
            ht = Meth->Payoffunc[i][j];
        }

        Meth->pLQij[i][j] = ht;

    }
}

/* printf("FIN de l'actualisation payoff de l'arbre des taux\ n");    */
}

static double OPTION(struct Tree *Meth)
{
    return Meth->pLQij[0][1];
}

```

```

}

static int DeleteTree(struct Tree *Meth)
{
    int i;

    for (i = 0; i < Meth->Ngrid + 1; i++)
    {
        free(Meth->pLRij[i]);
    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLQij[i]);
    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLPDo[i]);
    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLPMi[i]);
    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLPUp[i]);
    }
    for (i = 0; i < Meth->Ngrid; i++)
    {
        free(Meth->pLRef[i]);
    }

    free(Meth->pLRij);
    free(Meth->pLQij);
    free(Meth->pLPDo);
    free(Meth->pLPMi);
    free(Meth->pLPUp);
    free(Meth->pLRef);
    free(Meth->TSize);

    DeleteTimegrid(Meth);
    free(Meth->Payoffunc);
}

```

```

    return 1;
}

```

```

////////////////////////////////////
//////////////////////////////////// Specific functions of the product com
////////////////////////////////////

```

```

static void initPayoffSWAPTION(struct Tree *Meth, double T1, double per, double
{

```

```

    int i, j, k, n, N;
    double Swap0 = 0;
    double dZC0;
    double *cumul;

```

```

    /* Give the incice time for T1 the maturity of the option on the CB(T1,T1+per,
    /* Rk : T2 (S0 here), the final time of the tree is set to be final maturiy o
    n = indiceTime(Meth, T1);
    k = 1;
    N = Meth->TSize[n];

```

```

    /* computation of B(0,T1) */
    initPayoff1_tr(Meth, T1);
    Computepayoff(Meth, T1);

```

```

    dZC0 = Meth->pLQij[0][1];

```

```

    /* cumul[] will cumulate all the ZC BondPrice values, at index time n, compute
    cumul = malloc(N * sizeof(double));
    Meth->Payofffunc = malloc((n + 1) * sizeof(double *));

```

```

    for (i = 0; i < n + 1; i++)
    {
        Meth->Payofffunc[i] = malloc((Meth->TSize[i]) * sizeof(double));
    }

```

```

    /* Initialization of the variable tree payofffunc[] [] to zero */
    for (i = n; i >= 0; i--)

```

```

    {
        for (j = 0; j < Meth->TSize[i]; j++)
        {
            Meth->Payofffunc[i][j] = 0.;
        }
    }

    initPayoff1_tr(Meth, T2);
    Computepayoff(Meth, T2);

    /* So dZC0 will be dZC0=B(0,T1)-B(0,T2) */
    dZC0 = dZC0 - Meth->pLQij[0][1];

    /* cumul[] initialized with the last ZC BondPrice B(T1,T2) of the CBearing */
    for (j = 0; j < N; j++)
    {
        cumul[j] = (1 + K * per) * Meth->pLQij[n][j];
    }

    /* Computation of all B(T1, T1+k*per) of the CBearing and cumul in cumul[] */
    while (T1 + k * per < T2)
    {
        initPayoff1_tr(Meth, T1 + k * per);
        Computepayoff(Meth, T1 + k * per);

        for (j = 0; j < N; j++)
        {
            cumul[j] += K * per * Meth->pLQij[n][j]; /*pLQij[n][j] is, in the tree
        }
        Swap0 += per * Meth->pLQij[0][1];
        k = k + 1;
    }

    /* Initialization of the Payofffunc at the index time n of T1, thanks to cumul[]
    for (j = 0; j < N; j++)
    {
        Meth->Payofffunc[n][j] = cumul[j];
    }
    free(cumul);

    p->Par[0].Val.V_DOUBLE = 1.;

```

```

/*to define receiver or payer*/
for (j = 0; j < N; j++)
{
    Meth->Payoffunc[n][j] = (p->Compute)(p->Par, Meth->Payoffunc[n][j]);
}

/*Swap0 is the corresponding swap rate a time 0 */
Swap0 = dZC0 / Swap0;
/*printf("SwapATM:%lf\ n",Swap0);*/
}

/*Swaption=Option on Coupon-Bearing Bond*/
/*All details comments for the functions used here are mainly in "hwtree1dinclud
static int swaption_cirpp1d(int flat_flag, double a0, double b0, double t0, doub
{
    long Ns;
    double r0;
    ZCMarketData ZCMarket;

    a = a0;
    b = b0;
    rx0 = rc;
    sigma = sigma0;
    Ns = NtY * (long)(S0 / periodicity);
    Tr.Ngrid = Ns;

    r0 = rc;

    //N=(int)floor((S0-T0)/periodicity);

    /* Flag to decide to read or not ZC BondPrice datas in "initialyields.dat" */
    /* If B(0,T) not read then B(0,T)=exp(-FM*T) */
    /* If B(0,T) read then rcc becomes the futur knowing rate name here r0 */
    if (flat_flag == 0)
    {
        ZCMarket.FlatOrMarket = 0;
        ZCMarket.Rate = rc;
    }
}

```

```

else
{
    ZCMarket.FlatOrMarket = 1;
    ZCMarket.filename = curve;
    ReadMarketData(&ZCMarket);
}

/* S0 defines the final time tree variable (no time can be larger), Ns is the
SetTimegrid(&Tr, Ns, S0);
/* Allocate and initialize the tree*/
SetTree(&Tr);
/* translate the tree by "alpha" */
TranslateTree(&Tr, &ZCMarket);
/* Initialize the payoff for swaption as an option on CBearing(t=T0, T0, T0+pe
initPayoffSWAPTION(&Tr, T0, periodicity, S0, K, p); /*Comments of this functio
/* Compute the option from T0 to 0 in plQij[] [] tree variable */
Computepayoff(&Tr, T0);

/* return the result plQij[0][] or plQij[indiceTime(t0)][] in case of present
if (t0 == 0)
{
    *price = Nominal * OPTION(&Tr);
}
else
{
    *price = Nominal * OPTIONr_tr(&Tr, r0, t0);
}

/**delta=0;*/

DeleteTree(&Tr);

return OK;
}

int CALC(TR_SwaptionCIRpp1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

```

```

return swaption_cirpp1d(ptMod->flat_flag.Val.V_INT, ptMod->a.Val.V_DOUBLE, ptM
    ptMod->Sigma.Val.V_PDOUBLE, MOD(GetYield)(ptMod),
    MOD(GetCurve)(ptMod), ptOpt->BMaturity.Val.V_DATE,
    ptOpt->OMaturity.Val.V_DATE, ptOpt->PayOff.Val.V_NUMFU
    ptOpt->Nominal.Val.V_PDOUBLE, ptOpt->FixedRate.Val.V_P
    Met->Par[0].Val.V_LONG, &(Met->Res[0].Val.V_DOUBLE)/*,
}

```

```

static int CHK_OPT(TR_SwaptionCIRpp1D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PayerSwaption") == 0) || (strcmp(((Option
        return OK;
    else
        return WRONG;
}

```

```

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->Par[0].Val.V_LONG = 10;

    }
    return OK;
}

```

```

PricingMethod MET(TR_SwaptionCIRpp1D) =
{
    "TR_CIRpp1d_Swaption",
    { {"TimeStepNumber for Period", LONG, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_SwaptionCIRpp1D),
    {{"Price", DOUBLE, {100}, FORBID}/*,{"Delta",DOUBLE,{100},FORBID} */ , {" ", PR
    CHK_OPT(TR_SwaptionCIRpp1D),
    CHK_ok,
    MET(Init)
}

```

} ;