

[Help](#)

```
#include "cirpp1d_std.h"

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

static double *C;
static int nb_payement;

/*Shift function of the CIR++ model*/
static double shift(double a, double b, double sigma, double f0_s, double s)
{
    /* the shift rate of the cir++ model for x(0)=0 */
    double c;

    c = sqrt(a * a + 2 * sigma * sigma);

    return (f0_s - 2 * a * b * (exp(s * c) - 1) / (2 * c + (a + c) * (exp(s * c) -
}

static double A(double time, double a, double b, double sigma)
{
    double h = sqrt(SQR(a) + 2.*SQR(sigma));
    return pow(h * exp(0.5 * (a + h) * (time))) / (h + 0.5 * (a + h) * (exp(h * (ti
}

static double B(double time, double a, double b, double sigma)
{
    double h = sqrt(SQR(a) + 2.*SQR(sigma));
    return (exp(h * (time)) - 1.) / (h + 0.5 * (a + h) * (exp(h * (time)) - 1.));
}
```

```

/*Zero Coupon Bond*/
static double zcbond(double rcc, double a, double b, double sigma, double t, double T)
{
    if (t == 0)
    {
        return BondPrice(T, ZCMarket);
    }
    else
    {
        double h, A, B, At, AT, shift, c;
        double f0_t, P0_t, P0_T, P0_t_plus, P0_t_minus;

        P0_t = BondPrice(t, ZCMarket);
        P0_T = BondPrice(T, ZCMarket);

        /*Computation of Forward rate*/
        P0_t_plus = BondPrice(t * (1. + INC), ZCMarket);
        P0_t_minus = BondPrice(t * (1. - INC), ZCMarket);
        f0_t = -(log(P0_t_plus) - log(P0_t_minus)) / (2.*t * INC);

        /*A,B coefficient*/
        h = sqrt(SQR(a) + 2.*SQR(sigma));
        B = 2.*(exp(h * (T - t)) - 1.) / (2.*h + (a + h) * (exp(h * (T - t)) - 1.));
        A = pow(h * exp(0.5 * (a + h) * (T - t)) / (h + 0.5 * (a + h) * (exp(h * (T - t)) - 1.)), 2.);
        At = pow(h * exp(0.5 * (a + h) * (t)) / (h + 0.5 * (a + h) * (exp(h * (t)) - 1.)), 2.);
        AT = pow(h * exp(0.5 * (a + h) * (T)) / (h + 0.5 * (a + h) * (exp(h * (T)) - 1.)), 2.);

        c = sqrt(a * a + 2 * sigma * sigma);

        shift = (f0_t - 2 * a * b * (exp(t * c) - 1) / (2 * c + (a + c) * (exp(t * c) - 1)));

        A = A * (P0_T * At) / (AT * P0_t) * exp(B * shift);

        /*Price*/
        return A * exp(-B * rcc);
    }
}

/*Put Option on Zero Coupon Bond*/
static double zbput(double a, double b, double sigma, double rcc, double t, double T)
{

```

```

double PtS, PtT, ATS, BTS;
double f0_t;
double p1, p2, p3, k1, k2, k3, psi, phi, rb;
double h;

/*Computation of Forward rate*/
h = sqrt(SQR(a) + 2.*SQR(sigma));

if (t - 0.5 * INC > 0)
{
    f0_t = (log(BondPrice(t - 0.5 * INC, ZCMarket)) - log(BondPrice(t + 0.5 *
}
else
{
    f0_t = -log(BondPrice(INC, ZCMarket)) / INC;
}

PtT = zcbond(rcc, a, b, sigma, t, T, ZCMarket);
PtS = zcbond(rcc, a, b, sigma, t, S, ZCMarket);

BTS = B(S - T, a, b, sigma);
ATS = A(S - T, a, b, sigma);

/*X^2 parameters*/
rb = (log(ATS / K) + log(A(T, a, b, sigma) * BondPrice(S, ZCMarket)) - log(A(S
phi = 2.*h / (SQR(sigma) * (exp(h * (T - t)) - 1.));
psi = (a + h) / SQR(sigma);

p1 = 2.*rb * (phi + psi + BTS);
p2 = 4.*a * b / SQR(sigma);
p3 = 2.*SQR(phi) * (rcc - shift(a, b, sigma, f0_t, t)) * exp(h * (T - t)) / (p

k1 = 2.*rb * (phi + psi);
k2 = p2;
k3 = 2.*SQR(phi) * (rcc - shift(a, b, sigma, f0_t, t)) * exp(h * (T - t)) / (p

/*Price of Put by Parity*/

return PtS * pnl_cdfchi2n(p1, p2, p3) - K * PtT * pnl_cdfchi2n(k1, k2, k3) - P
}

```

```

/*Computation of Critical Rate*/
static double phi(double rr, double date, double periodicity, double first_payem
{
    int i;
    double sum, sum_der, ti, BB, maturity;

    sum = 0.;
    sum_der = 0.;
    maturity = first_payement - periodicity;

    for (i = 0; i <= nb_payement; i++)
    {
        ti = first_payement + (double)i * periodicity;
        BB = C[i] * zcbond(rr, a, b, sigma, maturity, ti, ZCMarket);
        sum += BB;
        sum_der += BB * (-B(ti - maturity, a, b, sigma));
    }

    return (sum - 1.) / sum_der;
}

static double Critical_Rate(double r, double date, double periodicity, double fi
{
    const double precision = 0.000001;
    double current = r;
    double error;

    do
    {
        error = phi(current, date, periodicity, first_payement, a, b, sigma, ZCMar
        current = current - error;
    }
    while (!(fabs(error) <= precision));
    return current;
}

/*Payer Swaption*/
static int ps_cirpp1d(int flat_flag, double a, double b, double date, double sig
{
    double sum, ti, new_K, critical_r, first_payement;
    int i;

```

```

ZCMarketData ZCMarket;

/* Flag to decide to read or not ZC bond datas in "initialyields.dat" */
/* If P(0,T) not read then P(0,T)=exp(-r0*T) */
if (flat_flag == 0)
{
    ZCMarket.FlatOrMarket = 0;
    ZCMarket.Rate = rcc;
}

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

first_payment = option_maturity + periodicity;
nb_payment = (int)((contract_maturity - first_payment) / periodicity);

/*Payer Swaption=Put Option on Coupon-Bearing Bond*/
C = malloc((nb_payment + 1) * sizeof(double));
for (i = 0; i < nb_payment; i++)
{
    C[i] = K * periodicity;
}
C[nb_payment] = 1 + K * periodicity;

/*Jamshidian decomposition*/
/*Computation of critical rate*/
critical_r = Critical_Rate(rcc, date, periodicity, first_payment, a, b, sigma);

/*Portfolio of CALL Option*/
sum = 0.;

for (i = 0; i <= nb_payment; i++)
{
    ti = first_payment + (double)i * periodicity;
    /*Strike*/

```

```

        new_K = zcbond(critical_r, a, b, sigma, option_maturity, ti, &ZCMarket);

        sum = sum + C[i] * zbput(a, b, sigma, rcc, date, option_maturity, ti, new_K);
    }
    /*Price*/
    *price = Nominal * sum;
    /*Delta*/
    /**delta=0.;*/

    free(C);
    return OK;
}

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

    return ps_cirpp1d(ptMod->flat_flag.Val.V_INT, ptMod->a.Val.V_DOUBLE, ptMod->b.Val.V_DOUBLE,
        ptMod->Sigma.Val.V_PDOUBLE, MOD(GetYield)(ptMod),
        MOD(GetCurve)(ptMod), ptOpt->Nominal.Val.V_PDOUBLE,
        ptOpt->FixedRate.Val.V_PDOUBLE, ptOpt->ResetPeriod.Val.V_DATE,
        ptOpt->BMaturity.Val.V_DATE, &(Met->Res[0].Val.V_DOUBLE)/*,&

}

static int CHK_OPT(CF_PayerSwaption)(void *Opt, void *Mod)
{
    return strcmp(((Option *)Opt)->Name, "PayerSwaption");
}

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

```

```
    return OK;
}
```

```
PricingMethod MET(CF_PayerSwaption) =
{
    "CF_Cirpp1d_PayerSwaption",
    {{ " ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(CF_PayerSwaption),
    {{ "Price", DOUBLE, {100}, FORBID} /*, {"Delta", DOUBLE, {100}, FORBID} */ , { " ", PR
    CHK_OPT(CF_PayerSwaption),
    CHK_ok,
    MET(Init)
} ;
```