

## [Help](#)

```
#include "
href../../../../mod/cirpp1d/cirpp1d_stdh/cirpp1d_stdh_src.pdfcirpp1d_stdh.h"
#include "pnl/pnl_vector.h"

//The "#else" part of the code will be freely available after the (year of creat
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2)
static int CHK_OPT(TR_ZBOCIRpp1D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_ZBOCIRpp1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/// TreeCIRpp1D      : structure that contains components of the tree (see TreeCI
/// ModelCIRpp1D     : structure that contains the parameters of the Hull&White
/// ZCMarketData : structure that contains the Zero Coupon Bond prices of the ma

static void ZCBond_InitialPayoffCIRpp1D(TreeCIRpp1D *Meth, PnlVect *ZCbondPriceV
{
    int NumberNode;

    NumberNode = (int)((GET(Meth->Xmax, Meth->Ngrid) - GET(Meth->Xmin, Meth->Ngrid

    pnl_vect_resize(ZCbondPriceVect, NumberNode + 1);

    pnl_vect_set_double(ZCbondPriceVect, 1.0); // Payoff = 1 for a ZC bond
}

/// Computation of the payoff at the final time of the tree (ie the option matur
static void ZCOption_InitialPayoffCIRpp1D(PnlVect *ZCbondPriceVect, PnlVect *Opt
{
    int j;

    double ZCPrice;
```

```

pnl_vect_resize(OptionPriceVect, ZCbondPriceVect->size);

/** Calcul du vecteur des payoffs a l'instant de maturite de l'option
for (j = 0 ; j < ZCbondPriceVect->size ; j++)
{
    ZCPrice = GET(ZCbondPriceVect, j);

    LET(OptionPriceVect, j) = (p->Compute)(p->Par, ZCPrice); // Payoff of the
}
}

void ZCOption_BackwardIterationCIRpp1D(TreeCIRpp1D *Meth, ModelCIRpp1D *ModelParam)
{
    double a, b, sigma;

    double delta_t, sqrt_delta_t;

    double current_rate, current_x, x_middle, ZCPrice;

    int i, h;
    int NumberNode, index;

    PnlVect *Probas;
    Probas = pnl_vect_create(3);

    /******* Model parameters *****/
    a = (ModelParam->MeanReversion);
    b = (ModelParam->LongTermMean);
    sigma = (ModelParam->Volatility);

    delta_t = GET(Meth->t, 1) - GET(Meth->t, 0); // = t[i] - t[i-1]
    sqrt_delta_t = sqrt(delta_t);

    for (i = index_last - 1; i >= index_first; i--)
    {
        NumberNode = (int)((GET(Meth->Xmax, i) - GET(Meth->Xmin, i)) / (Meth->delta_t));

        pnl_vect_resize(OptionPriceVect1, NumberNode + 1); // OptionPriceVect1 :=

```

```

if (Eur_Or_Am != 0)
{
    pnl_vect_resize(ZCbondPriceVect1, NumberNode + 1); // OptionPrice1 :=
}

// Loop over the node at the time i
for (h = 0 ; h <= NumberNode ; h++)
{
    current_x = x_value(i, h, Meth);
    current_rate = R(current_x, sigma) + GET(Meth->alpha, i);

    x_middle = MiddleNode(Meth, i, a, b, sigma, current_x, sqrt_delta_t, P

    index = (int)((x_middle - GET(Meth->Xmin, i + 1)) / (Meth->delta_x) +

    LET(OptionPriceVect1, h) = exp(-current_rate * delta_t) * (GET(Probas,

    if (Eur_Or_Am != 0)
    {
        LET(ZCbondPriceVect1, h) = exp(-current_rate * delta_t) * (GET(Pro

        ZCPrice = GET(ZCbondPriceVect1, h); // ZC price P(ti, S, r_ti=curren
        // In the case of american option, decide wether to exerice the op
        if (GET(OptionPriceVect1, h) < (p->Compute)(p->Par, ZCPrice))
        {
            LET(OptionPriceVect1, h) = (p->Compute)(p->Par, ZCPrice);
        }
    }
}

pnl_vect_clone(OptionPriceVect2, OptionPriceVect1); // Copy OptionPriceVec
if (Eur_Or_Am != 0)
{
    pnl_vect_clone(ZCbondPriceVect2, ZCbondPriceVect1);
}

} // END of the loop on i (time)

pnl_vect_free(&Probas);
}

```

```

/// Prix at time s of an option, maturing at T, on a ZC, with maturity S, using
double tr_cirpp1d_zcoption(TreeCIRpp1D *Meth, ModelCIRpp1D *ModelParam, ZCMarket
{
    double sigma;
    double delta_t, delta_r;
    double current_rate, current_x;

    double theta, OptionPrice1, OptionPrice2;
    double OptionPrice;
    int i_s, i_T;
    int j_r;

    PnlVect *Probas;
    PnlVect *OptionPriceVect1; // Matrix of prices of the option at i
    PnlVect *OptionPriceVect2; // Matrix of prices of the option at i+1
    PnlVect *ZCbondPriceVect1; // Vector of prices of the option at time i
    PnlVect *ZCbondPriceVect2; // Vector of prices of the option at time i+1

    Probas = pnl_vect_create(3);
    OptionPriceVect1 = pnl_vect_create(1);
    OptionPriceVect2 = pnl_vect_create(1);
    ZCbondPriceVect1 = pnl_vect_create(1);
    ZCbondPriceVect2 = pnl_vect_create(1);

    ///***** Model parameters *****/
    //a = (ModelParam->MeanReversion);
    //b = (ModelParam->LongTermMean);
    sigma = (ModelParam->Volatility);
    current_x = ModelParam->Initialx0; // x(0)

    delta_t = GET(Meth->t, 1) - GET(Meth->t, 0); // = t[i] - t[i-1]
    //sqrt_delta_t = sqrt(delta_t);

    ///***** Computation of the vector of payoff at the maturity of t
    i_T = indiceTimeCIRpp1D(Meth, T); // Localisation of s on the tree

    ZCBond_InitialPayoffCIRpp1D(Meth, ZCbondPriceVect2);

    ZCOption_BackwardIterationCIRpp1D(Meth, ModelParam, ZCbondPriceVect1, ZCbondPr

    ZCOption_InitialPayoffCIRpp1D(ZCbondPriceVect2, OptionPriceVect2, p);

```

```

//***** Backward computation of the option price until initial t
i_s = indiceTimeCIRpp1D(Meth, s); // Localisation of s on the tree

if (i_s == 0) // If s=0
{
    ZCOption_BackwardIterationCIRpp1D(Meth, ModelParam, ZCbondPriceVect1, ZCbo

    current_rate = R(current_x, sigma) + GET(Meth->alpha, 0);

    OptionPrice = exp(-current_rate * delta_t) * (GET(Probas, 2) * GET(OptionP

}
else
{
    // We compute the price of the option as a linear interpolation of the pri

    j_r = (int)((2 * sqrt(r - GET(Meth->alpha, i_s)) / sigma - GET(Meth->Xmin,

    if (j_r < 0 || j_r > (GET(Meth->Xmax, i_s) - GET(Meth->Xmin, i_s)) / (Meth
    {
        printf("WARNING : Instantaneous futur spot rate is out of tree\ n");
        exit(EXIT_FAILURE);
    }

    ZCOption_BackwardIterationCIRpp1D(Meth, ModelParam, ZCbondPriceVect1, ZCbo

    current_x = x_value(i_s, j_r, Meth);

    current_rate = R(current_x, sigma) + GET(Meth->alpha, i_s);

    delta_r = R(x_value(i_s, j_r + 1, Meth), sigma) - current_x;

    theta = (r - current_rate) / delta_r ;

    OptionPrice1 = GET(OptionPriceVect1, j_r);

    OptionPrice2 = GET(OptionPriceVect1, j_r + 1);

    OptionPrice = (1 - theta) * OptionPrice1 + theta * OptionPrice2 ;

```

```

    }

    pnl_vect_free(& OptionPriceVect1);
    pnl_vect_free(& OptionPriceVect2);
    pnl_vect_free(& ZCbondPriceVect1);
    pnl_vect_free(& ZCbondPriceVect2);
    pnl_vect_free(& Probas);

    return OptionPrice;

} // FIN de la fonction ZCOption

static int tr_zbold(int flat_flag, double t, double r0, char *curve, double a, d
{
    TreeCIRpp1D Tr;
    ModelCIRpp1D ModelParams;
    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 = r0;
    }

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

        if (T > GET(ZCMarket.tm, ZCMarket.Nvalue - 1))
        {
            printf("\ nError : time bigger than the last time value entered in ini
            exit(EXIT_FAILURE);
        }
    }
}

```

```

ModelParams.MeanReversion = a;
ModelParams.LongTermMean  = b;
ModelParams.Volatility     = sigma;
ModelParams.Initialx0      = 2;

SetTimegridZCbondCIRpp1D(&Tr, N_steps, t, T, S);

SetTreeCIRpp1D(&Tr, &ModelParams, &ZCMarket);

//Price of an option on a ZC
*price = tr_cirpp1d_zcoption(&Tr, &ModelParams, &ZCMarket, T, S, p, t, r0, am

DeleteTreeCIRpp1D(&Tr);
DeleteZCMarketData(&ZCMarket);

return OK;
}

//***** PREMIA FUNCTIONS *****
int CALC(TR_ZBOCIRpp1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return tr_zbo1d(ptMod->flat_flag.Val.V_INT,
                    ptMod->T.Val.V_DATE,
                    MOD(GetYield)(ptMod),
                    MOD(GetCurve)(ptMod),
                    ptMod->a.Val.V_DOUBLE,
                    ptMod->b.Val.V_DOUBLE,
                    ptMod->Sigma.Val.V_PDOUBLE,
                    ptOpt->BMaturity.Val.V_DATE,
                    ptOpt->OMaturity.Val.V_DATE,
                    ptOpt->PayOff.Val.V_NUMFUNC_1,
                    ptOpt->EuOrAm.Val.V_BOOL,
                    Met->Par[0].Val.V_LONG,
                    &(Met->Res[0].Val.V_DOUBLE));
}

```

```

}

static int CHK_OPT(TR_ZBOCIRpp1D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "ZeroCouponCallBondEuro") == 0) || (strcmp(
        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 = 200;
    }
    return OK;
}

PricingMethod MET(TR_ZBOCIRpp1D) =
{
    "TR_CIRpp1D_ZBO",
    { {"TimeStepNumber", LONG, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_ZBOCIRpp1D),
    {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(TR_ZBOCIRpp1D),
    CHK_ok,
    MET(Init)
} ;

```