

[Help](#)

```
#include <stdlib.h>
#include "
href../../../../mod/lrshjm1d/lrshjm1d_std/lrshjm1d_std_h_src.pdf/lrshjm1d_std.h"
#include "
href../../../../common/math/InterestRateModelTree/TreeLRS1D/TreeLRS1D_h_src.pdf/math
#include "pnl/pnl_vector.h"
#include "
href../../../../common/math/read_market_zc/InitialYieldCurve_h_src.pdf/math/read_mar

//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_SwaptionLRS1D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_SwaptionLRS1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/// TreeLRS1D      : structure that contains components of the tree (see TreeLRS1
/// ModelLRS1D     : structure that contains the parameters of the Hull&White on
/// ZCMarketData   : structure that contains the Zero Coupon Bond prices of the ma

static double cf_lrs1d_zcb(ZCMarketData *ZCMarket, double t, double r0, double p
{
    if (t == 0)
    {
        return BondPrice(T, ZCMarket);
    }
    else
    {
        double price;
        double P0_t, P0_T, P0_t_plus, P0_t_minus, f0_t, CapitalLambda;
        double dt;
```

```

    CapitalLambda = (1 - exp(-kappa * (T - t))) / kappa;

    dt = INC * t;

    PO_t = BondPrice(t, ZCMarket);

    PO_T = BondPrice(T, ZCMarket);

    PO_t_plus = BondPrice(t + dt, ZCMarket);

    PO_t_minus = BondPrice(t - dt, ZCMarket);

    f0_t = -(log(PO_t_plus) - log(PO_t_minus)) / (2 * dt);

    //Price of Zero Coupon Bond
    price = (PO_T / PO_t) * exp(-SQR(CapitalLambda) * phi0 / 2 + CapitalLambda

    return price;
}

}

/// Computation of the payoff at the final time of the tree (ie the option matur
void Swaption_InitialPayoffLRS1D(TreeLRS1D *Meth, ModelLRS1D *ModelParam, ZCMark
{
    double sigma, rho, kappa, lambda;

    int i, j, h, NumberOfPayments;
    double delta_y, delta_t, sqrt_delta_t;
    double y_00, y_ih, r_ih, phi_ihj;

    double Ti, ZCPrice, SumZC;

    /// Model Parameters
    kappa = (ModelParam->Kappa);
    sigma = (ModelParam->Sigma);
    rho = (ModelParam->Rho);
    lambda = (ModelParam->Lambda);

    ZCPrice = 0.;
    pnl_vect_resize(OptionPriceVect2, 6 * (Meth->Ngrid) - 3);

```

```

delta_t = GET(Meth->t, 1) - GET(Meth->t, 0);
sqrt_delta_t = sqrt(delta_t);
delta_y = lambda * sqrt_delta_t;

y_00 = r_to_y(ModelParam, -log(BondPrice(GET(Meth->t, 1), ZCMarket)) / delta_t

NumberOfPayments = (int) floor((contract_maturity - option_maturity) / periodi

p->Par[0].Val.V_DOUBLE = 1.0;

for (h = 0; h <= 2 * (Meth->Ngrid); h++) /// h : numero de la box
{
    y_ih = y_00 + ((Meth->Ngrid) - h) * delta_y;
    r_ih = y_to_r(ModelParam, y_ih);

    for (j = 0; j < number_phi_in_box(Meth->Ngrid, h); j++) /// Boucle sur les
    {
        phi_ihj = phi_value(Meth, Meth->Ngrid, h, j);

        SumZC = 0;
        for (i = 1; i <= NumberOfPayments; i++)
        {
            Ti = option_maturity + i * periodicity;
            ZCPrice = cf_lrs1d_zcb(ZCMarket, option_maturity, r_ih, phi_ihj, k

            SumZC += ZCPrice;
        }

        //SwapRate = (1-ZCPrice) / (periodicity*SumZC);

        LET(OptionPriceVect2, index_tree(Meth->Ngrid , h, j)) = ((p->Compute)(
    }
}

}

/// Backward computation of the price
void Swaption_BackwardIterationLRS1D(TreeLRS1D *Meth, ModelLRS1D *ModelParam, ZC
{
    double sigma, rho, kappa, lambda;

```

```

int i, j, h;
double delta_y, delta_t, sqrt_delta_t;
double price_up, price_middle, price_down;
double y_00, y_ih, r_ih, phi_ihj, phi_next;

PnlVect *proba_from_ij;

proba_from_ij = pnl_vect_create(3);

///<***** Model parameters *****/
kappa = (ModelParam->Kappa);
sigma = (ModelParam->Sigma);
rho = (ModelParam->Rho);
lambda = (ModelParam->Lambda);

delta_t = GET(Meth->t, 1) - GET(Meth->t, 0);
y_00 = r_to_y(ModelParam, -log(BondPrice(GET(Meth->t, 1), ZCMarket)) / delta_t);

for (i = index_last - 1; i >= index_first; i--)
{
    pnl_vect_resize(OptionPriceVect1, 6 * i - 3); // OptionPriceVect1 := Price

    delta_t = GET(Meth->t, i + 1) - GET(Meth->t, i);
    sqrt_delta_t = sqrt(delta_t);
    delta_y = lambda * sqrt_delta_t;

    for (h = 0; h <= 2 * i; h++) /// h : numero de la box
    {
        y_ih = y_00 + (i - h) * delta_y;
        r_ih = y_to_r(ModelParam, y_ih);

        for (j = 0; j < number_phi_in_box(i, h); j++) /// Boucle sur les vale
        {
            phi_ihj = phi_value(Meth, i, h, j);

            phi_next = phi_ihj * (1 - 2 * kappa * delta_t) + SQR(sigma) * pow(

            price_up      = Interpolation(Meth, i + 1, h , OptionPriceVect2, p

```

```

        price_middle = Interpolation(Meth, i + 1, h + 1, OptionPriceVect2,
        price_down    = Interpolation(Meth, i + 1, h + 2, OptionPriceVect2,

        probabilities(GET(Meth->t, i), y_ih, phi_ihj, lambda, sqrt_delta_t

        LET(OptionPriceVect1, index_tree(i, h, j)) = exp(-r_ih * delta_t)

    }
}

pnl_vect_clone(OptionPriceVect2, OptionPriceVect1); // Copy OptionPriceVec

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

pnl_vect_free(&proba_from_ij);
}

/// Price of a swaption using a trinomial tree
double tr_lrs1d_swaption(TreeLRS1D *Meth, ModelLRS1D *ModelParam, ZCMarketData *
{
    double lambda;

    double delta_y; // delta_x1 = space step of the process x at time i ; delta_x2
    double delta_t, sqrt_delta_t; // time step

    double OptionPrice, OptionPrice1, OptionPrice2;
    int i_s, h_r;
    double theta;
    double y_r, y_ih, y_00, r_00;

    PnlVect *proba_from_ih;
    PnlVect *OptionPriceVect1; // Matrix of prices of the option at i
    PnlVect *OptionPriceVect2; // Matrix of prices of the option at i+1

    proba_from_ih = pnl_vect_create(3);
    OptionPriceVect1 = pnl_vect_create(1);
    OptionPriceVect2 = pnl_vect_create(1);

    ///***** Model parameters *****/
    lambda = (ModelParam->Lambda);

```

```

//***** Computation of the vector of payoff at the maturity of t
Swaption_InitialPayoffLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVect2, p, p

//***** Backward computation of the option price until initial t
i_s = indiceTimeLRS1D(Meth, s); // Localisation of s on the tree

delta_t = GET(Meth->t, 1) - GET(Meth->t, 0);
sqrt_delta_t = sqrt(delta_t);

r_00 = -log(BondPrice(GET(Meth->t, 1), ZCMarket)) / delta_t;
y_00 = r_to_y(ModelParam, r_00);

if (i_s == 0) // If s=0
{
    Swaption_BackwardIterationLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVec
    probabilities(GET(Meth->t, 0), y_00, 0, lambda, sqrt_delta_t, ModelParam,
    OptionPrice = exp(-r_00 * delta_t) * (GET(proba_from_ih, 0) * GET(OptionPr
}

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

    delta_t = GET(Meth->t, i_s + 1) - GET(Meth->t, i_s);
    sqrt_delta_t = sqrt(delta_t);
    delta_y = lambda * sqrt_delta_t;

    y_r = r_to_y(ModelParam, r);

    h_r = (int) floor(i_s - (y_r - y_00) / delta_y); // y_r between y(h_r) et

    y_ih = y_00 + (i_s - h_r) * delta_y;

    if (h_r < 0 || h_r > 2 * i_s)
    {
        printf("WARNING : Instantaneous futur spot rate is out of tree\ n");
        exit(EXIT_FAILURE);
    }
}

```

```

    Swaption_BackwardIterationLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVect1, OptionPriceVect2);

    theta = (y_ih - y_r) / delta_y;

    OptionPrice1 = MeanPrice(Meth, i_s, h_r, OptionPriceVect2); //Interpolation
    OptionPrice2 = MeanPrice(Meth, i_s, h_r + 1, OptionPriceVect2); // Interpolation

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

pnl_vect_free(& OptionPriceVect1);
pnl_vect_free(& OptionPriceVect2);
pnl_vect_free(&proba_from_ih);

return OptionPrice;

}

```

```

static int tr_swaption1d(int flat_flag, double t, double r0, char *curve, double *pnl_vect)
{
    TreeLRS1D Tr;
    ModelLRS1D 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 (option_maturity > GET(ZCMarket.tm, ZCMarket.Nvalue - 1))
        {
            printf("\ nError : time bigger than the last time value entered in ini
            exit(EXIT_FAILURE);
        }
    }

    ModelParams.Kappa = kappa;
    ModelParams.Sigma = sigma;
    ModelParams.Rho = rho;
    ModelParams.Lambda = lambda;

    // Construction of the Time Grid
    SetTimegridLRS1D(&Tr, N_steps, t, option_maturity);

    // Construction of the tree, calibrated to the initial yield curve
    SetTreeLRS1D(&Tr, &ModelParams, &ZCMarket);

    *price = Nominal * tr_lrs1d_swaption(&Tr, &ModelParams, &ZCMarket, N_steps, p,

    DeleteTreeLRS1D(&Tr);
    DeleteZCMarketData(&ZCMarket);

    return OK;
}

///  

//***** PREMIA FUNCTIONS *****  

int CALC(TR_SwaptionLRS1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return tr_swaption1d(ptMod->flat_flag.Val.V_INT,
                        ptMod->T.Val.V_DATE,
                        MOD(GetYield)(ptMod),
                        MOD(GetCurve)(ptMod),
                        ptMod->Kappa.Val.V_DOUBLE,
                        ptMod->Sigma.Val.V_PDOUBLE,
                        ptMod->Rho.Val.V_PDOUBLE,

```



```

        ptMod->Lambda.Val.V_PDOUBLE,
        ptOpt->BMaturity.Val.V_DATE,
        ptOpt->OMaturity.Val.V_DATE,
        ptOpt->ResetPeriod.Val.V_DATE,
        ptOpt->Nominal.Val.V_PDOUBLE,
        ptOpt->FixedRate.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_1,
        Met->Par[0].Val.V_LONG,
        &(Met->Res[0].Val.V_DOUBLE));
}
static int CHK_OPT(TR_SwaptionLRS1D)(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 = 200;
    }

    return OK;
}

PricingMethod MET(TR_SwaptionLRS1D) =
{
    "TR_LRS1D_Swaption",
    { {"TimeStepNumber", LONG, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_SwaptionLRS1D),
    {"Price", DOUBLE, {100}, FORBID}/*,{"Delta",DOUBLE,{100},FORBID}*/ , {" ", PR
    CHK_OPT(TR_SwaptionLRS1D),
    CHK_ok,

```

```
    MET(Init)
} ;
```