

Help

```
#include <stdlib.h>
#include "lrshjmid_std.h"
#include "math/InterestRateModelTree/TreeLRS1D/TreeLRS1D.h"
#include "pnl/pnl_vector.h"
#include "math/read_market_zc/InitialYieldCurve.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(TR_CapFloorLRS1D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_CapFloorLRS1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

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;

        P0_t = BondPrice(t, ZCMarket);

        P0_T = BondPrice(T, ZCMarket);

        P0_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
static void CapFloor_InitialPayoffLRS1D(TreeLRS1D *Meth, ModelLRS1D *ModelParam,
{
    double sigma, rho, kappa, lambda;

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

    double ZCPrice;

    int i_T1;
    double periodicity;

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

    /// Computation of the vector of payoff at the maturity of the option
    periodicity = T2 - T1;
    i_T1 = indiceTimeLRS1D(Meth, T1);

    pnl_vect_resize(OptionPriceVect2, 6 * i_T1 - 3);

    delta_t = GET(Meth->t, i_T1 + 1) - GET(Meth->t, i_T1);
    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)) / GET(Met

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

for (h = 0; h <= 2 * i_T1; h++) /// h : numero de la box
{
    y_ih = y_00 + (i_T1 - h) * delta_y;
    r_ih = y_to_r(ModelParam, y_ih);

    for (j = 0; j < number_phi_in_box(i_T1, h); j++) /// Boucle sur les valeurs
    {
        phi_ihj = phi_value(Meth, i_T1, h, j);

        ZCPrice = cf_lrs1d_zcb(ZCMarket, T1, r_ih, phi_ihj, kappa, sigma, rho,

        LET(OptionPriceVect2, index_tree(i_T1, h, j)) = (p->Compute)(p->Par, (

    }
}

}

/// Backward computation of the price of a Zero Coupon Bond
static void CapFloor_BackwardIterationLRS1D(TreeLRS1D *Meth, ModelLRS1D *ModelPa
{
    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 valeurs
        {
            phi_ihj = phi_value(Meth, i, h, j);

            phi_next = phi_ihj * (1 - 2 * kappa * delta_t) + SQR(sigma) * pow(
                GET(Meth->t, i + 1) - GET(Meth->t, i), 0.5);

            price_up = Interpolation(Meth, i + 1, h, OptionPriceVect2, p);
            price_middle = Interpolation(Meth, i + 1, h + 1, OptionPriceVect2, p);
            price_down = Interpolation(Meth, i + 1, h + 2, OptionPriceVect2, p);

            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 OptionPriceVect2 to OptionPriceVect1
} // END of the loop on i (time)

```

```

    pnl_vect_free(&proba_from_ij);

}

/// Price of a Cap/Floor using a trinomial tree

static double tr_lrs1d_capfloor(TreeLRS1D *Meth, ModelLRS1D *ModelParam, ZCMarket
{
    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, i_s, h_r;
    double theta;
    double y_r, y_ih, y_00, r_00;

    double Ti2, Ti1;
    int i_Ti2, i_Ti1, n;

    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);

    ///***** PAYOFF at the MATURITY of the OPTION : T(n-1)*****
    Ti2 = contract_maturity;
    Ti1 = Ti2 - periodicity;

    CapFloor_InitialPayoffLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVect2, p, T

    ///***** Backward computation of the option price *****/

```

```

n = (int)((contract_maturity - first_reset_date) / periodicity + 0.1);

if (n > 1)
{
    for (i = n - 2; i >= 0; i--)
    {
        Ti1 = first_reset_date + i * periodicity;
        Ti2 = Ti1 + periodicity;
        i_Ti2 = indiceTimeLRS1D(Meth, Ti2);
        i_Ti1 = indiceTimeLRS1D(Meth, Ti1);

        CapFloor_BackwardIterationLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVec, i_Ti2, i_Ti1);

        CapFloor_InitialPayoffLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVect1, i_Ti1);

        pnl_vect_plus_vect(OptionPriceVect2, OptionPriceVect1);
    }
}

///  

//***** Price of the option at initial time s *****  

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);

Ti1 = first_reset_date;
i_Ti1 = indiceTimeLRS1D(Meth, Ti1);

if (i_s == 0) // If s=0
{
    CapFloor_BackwardIterationLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVect1, i_Ti1, i_s);

    probabilities(GET(Meth->t, 0), y_00, 0, lambda, sqrt_delta_t, ModelParam, i_Ti1);

    OptionPrice = exp(-r_00 * delta_t) * (GET(proba_from_ih, 0) * GET(OptionPriceVect1, i_Ti1));
}
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);
    }

    CapFloor_BackwardIterationLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVec

    theta = (y_ih - y_r) / delta_y;

    OptionPrice1 = MeanPrice(Meth, i_s, h_r, OptionPriceVect2); //Interpolatio

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

    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_capfloor1d(int flat_flag, double t, double r0, char *curve, double

```

```

{
    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 (contract_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
    SetTimegridCapLRS1D(&Tr , NtY, t, first_reset_date, contract_maturity - period

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

    *price = Nominal * tr_lrs1d_capfloor(&Tr, &ModelParams, &ZCMarket, NtY, p, t,

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

```



```

    return OK;
}

```

```

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

```

```

int CALC(TR_CapFloorLRS1D)(void *Opt, void *Mod, PricingMethod *Met)
{

```

```

    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

```

```

    return tr_capfloor1d(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->FirstResetDate.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_CapFloorLRS1D)(void *Opt, void *Mod)
{

```

```

    if ((strcmp(((Option *)Opt)->Name, "Cap") == 0) || (strcmp(((Option *)Opt)->Name, "Floor") == 0))
        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_CapFloorLRS1D) =
{
    "TR_LRS1D_CapFloor",
    { {"TimeStepNumber for Period", LONG, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_CapFloorLRS1D),
    {{"Price", DOUBLE, {100}, FORBID}/*,{"Delta",DOUBLE,{100},FORBID} */, {" ", PR
    CHK_OPT(TR_CapFloorLRS1D),
    CHK_ok,
    MET(Init)
} ;

```