```
   Help
#include <stdlib.h>
#include "lrshjm1d_stdi.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 "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(TR_CapFloorLRS1D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_CapFloorLRS1D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double cf_lrs1d_zcb(ZCMarketData* ZCMarket, double
    t, double r0, double phi0, double kappa, double sigma,
    double rho, double lambda, double T)
{
    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);

        P0_t_minus = BondPrice(t - dt, ZCMarket);

        f0_t = -(log(P0_t_plus) - log(P0_t_minus))/ (2 * dt
    );

        //Price of Zero Coupon Bond
        price = (P0_T/P0_t) * exp(-SQR(CapitalLambda)*phi0/
    2 + CapitalLambda*(f0_t-r0));

        return price;
    }

}


/// Computation of the payoff at the final time of the tre
    e (ie the option maturity)
static void CapFloor_InitialPayoffLRS1D(TreeLRS1D* Meth,
    ModelLRS1D* ModelParam, ZCMarketData* ZCMarket, PnlVect*
    OptionPriceVect2, NumFunc_1 *p, double T1, double T2, double     CapFloorFix
{
    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 maturit
    y 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(Meth->t, 1));

    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++) /// Bouc
    le sur les  valeurs de phi à (i,h)
        {
            phi_ihj = phi_value(Meth, i_T1, h, j);

            ZCPrice = cf_lrs1d_zcb(ZCMarket, T1, r_ih, phi_
    ihj, kappa, sigma, rho, lambda, T2);

            LET(OptionPriceVect2, index_tree(i_T1, h, j)) =
     (p->Compute)(p->Par, (1+periodicity*CapFloorFixedRate)*
    ZCPrice);

        }
    }

}

/// Backward computation of the price of a Zero Coupon Bond
static void CapFloor_BackwardIterationLRS1D(TreeLRS1D*
```

```
    Meth, ModelLRS1D* ModelParam, ZCMarketData* ZCMarket, PnlVect*
     OptionPriceVect1, PnlVect* OptionPriceVect2, int index_
    last, int index_first)
{
    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 of the bond in the tree at time t(i)

        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++) /// Bouc
    le sur les  valeurs de phi à (i,h)
            {
                phi_ihj = phi_value(Meth, i, h, j);

                phi_next = phi_ihj * (1-2*kappa*delta_t) +
    SQR(sigma) * pow(y_to_r(ModelParam, y_ih), (2*rho)) * delt
    a_t;

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

                probabilities(GET(Meth->t,i), y_ih, phi_ih
    j, lambda, sqrt_delta_t, ModelParam, ZCMarket, proba_from_
    ij);

                LET(OptionPriceVect1, index_tree(i,h,j)) =
    exp(-r_ih*delta_t) * (GET(proba_from_ij,0) * price_up + GET(
    proba_from_ij,1) * price_middle + GET(proba_from_ij,2) *
    price_down );

            }
        }

        pnl_vect_clone(OptionPriceVect2, OptionPriceVect1);
     // Copy OptionPriceVect1 in OptionPriceVect2

    } // 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, ModelLRS1
    D* ModelParam, ZCMarketData* ZCMarket, int NumberOfTimeStep
```

```
    , NumFunc_1 *p, double s, double r, double periodicity,
    double first_reset_date,double contract_maturity, double CapF
    loorFixedRate)
{
    double lambda;

    double delta_y; // delta_x1 = space step of the proces
    s x at time i ; delta_x2 same at time i+1.
    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, Ti1, Ti2, CapFloorFixedRate);


    ///**************** Backward computation of the option
```

```
price ****************///
n = (int) ((contract_maturity-first_reset_date)/perio
dicity + 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, ModelPar
am, ZCMarket, OptionPriceVect1, OptionPriceVect2, i_Ti2, i_
Ti1);

        CapFloor_InitialPayoffLRS1D(Meth, ModelParam,
ZCMarket, OptionPriceVect1, p, Ti1, Ti2, CapFloorFixedRate);

        pnl_vect_plus_vect(OptionPriceVect2, OptionPric
eVect1);
    }
}

///****************** 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))/delt
a_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, OptionPriceVect2, i_Ti1, 1);

    probabilities(GET(Meth->t,0), y_00, 0, lambda, sq
rt_delta_t, ModelParam, ZCMarket, proba_from_ih);

    OptionPrice = exp(-r_00*delta_t) * ( GET(proba_fro
m_ih,0) * GET(OptionPriceVect1, 0) + GET(proba_from_ih,1) *
 GET(OptionPriceVect1,1) + GET(proba_from_ih,2) * GET(
OptionPriceVect1, 2));
}

else
{   // We compute the price of the option as a linear
interpolation of the prices at the nodes r(i_s,j_r) and r(i_s,
j_r+1)

    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(h_r+1) : y(h_r+1) < y_r <= y(h_r)

    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, OptionPriceVect1, OptionPriceVect2, i_Ti1, i_s);

    theta = (y_ih - y_r)/delta_y;
```

```
        OptionPrice1 = MeanPrice(Meth, i_s, h_r, OptionPric
    eVect2); //Interpolation(Meth, i_s, h_r  , OptionPriceVect2
    , phi0);

        OptionPrice2 = MeanPrice(Meth, i_s, h_r+1, OptionP
    riceVect2); // Interpolation(Meth, i_s, h_r+1  , OptionPric
    eVect2, phi0);

        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,
    double kappa, double sigma, double rho, double lambda, double
    contract_maturity, double first_reset_date, double periodicity
    ,double Nominal, double CapFloorFixedRate, NumFunc_1 *p,
    long NtY, double *price)
{
    TreeLRS1D Tr;
    ModelLRS1D ModelParams;
    ZCMarketData ZCMarket;

    /* Flag to decide to read or not ZC bond datas in "ini
    tialyields.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;
        ReadMarketData(&ZCMarket);

        if(contract_maturity > GET(ZCMarket.tm,ZCMarket.Nv
    alue-1))
        {
            printf("{nError : time bigger than the last
    time value entered in initialyield.dat{n");
            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-periodicity, periodicity); // TimeGride fro
    m 0 to T(n-1)=contract_maturity-periodicity

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

    *price =  Nominal * tr_lrs1d_capfloor(&Tr, &ModelParam
    s, &ZCMarket, NtY, p, t, r0, periodicity, first_reset_date,
     contract_maturity, CapFloorFixedRate);

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

    return OK;
}



///********************************************** PREMIA
```

```
    FUNCTIONS ************************************************///

int CALC(TR_CapFloorLRS1D)(void *Opt,void *Mod,Pricing
    Method *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),
                            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_DA
    TE,
                            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},FO
    RBID} */,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_CapFloorLRS1D),
  CHK_ok,
  MET(Init)
} ;
```

References