Help

```
#include "lrshjm1d_stdi.h"
#include "math/InterestRateModelTree/TreeLRS1D/TreeLRS1D.h"
#include "pnl/pnl_vector.h"
#include "math/read_market_zc/InitialYieldCurve.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_ZCBondLRS1D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_ZCBondLRS1D)(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

/// Computation of the payoff at the final time of the tree (ie the ZCBond matur
static void ZCBond_InitialPayoffLRS1D(TreeLRS1D *Meth, PnlVect *OptionPriceVect2
{
  pnl_vect_resize(OptionPriceVect2, 6 * (Meth->Ngrid) - 3);

  pnl_vect_set_double(OptionPriceVect2, 1.0); // Payoff = 1 for a ZC bond
}

/// Backward computation of the price of a Zero Coupon Bond
static void ZCBond_BackwardIterationLRS1D(TreeLRS1D *Meth, ModelLRS1D *ModelPara
{
  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 at time "s" of a ZC bond maturing at "T" using a trinomial tree.
static double tr_lrs1d_zcbond(TreeLRS1D *Meth, ModelLRS1D *ModelParam, ZCMarketD
{
  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
  ZCBond_InitialPayoffLRS1D(Meth, OptionPriceVect2);
```

```
///****************** Backward computation of the option price until 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);

if (i_s == 0) // If s=0
  {
    ZCBond_BackwardIterationLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVect1

    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);
      }

    ZCBond_BackwardIterationLRS1D(Meth, ModelParam, ZCMarket, OptionPriceVect1
```

```
      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_zcbond1d(int flat_flag, double t, double r0, char *curve, double k
{
  TreeLRS1D Tr;
  ModelLRS1D ModelParams;
  ZCMarketData ZCMarket;

  //N_steps = 300;
  //T = 6;

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

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

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

  //Price of Zero Coupon Bond
  *price = tr_lrs1d_zcbond(&Tr, &ModelParams, &ZCMarket, T, t, r0);

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

  return OK;
}



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

int CALC(TR_ZCBondLRS1D)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  return tr_zcbond1d(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,
                      Met->Par[0].Val.V_LONG,
                      &(Met->Res[0].Val.V_DOUBLE));
}
static int CHK_OPT(TR_ZCBondLRS1D)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "ZeroCouponBond") == 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 = 100;
    }
  return OK;
}

PricingMethod MET(TR_ZCBondLRS1D) =
{
  "TR_LRS1D_ZCBond",
  { {"TimeStepNumber", LONG, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(TR_ZCBondLRS1D),
  {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(TR_ZCBondLRS1D),
  CHK_ok,
  MET(Init)
} ;
```