Help

```
#include <stdlib.h>
#include "blackkarasinski1d_stdi.h"
#include "pnl/pnl_vector.h"
#include "math/InterestRateModelTree/TreeShortRate/TreeShortRate.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_ZBOBK1D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_ZBOBK1D)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/// TreeShortRate     : structure that contains components of the tree (see Tree
/// ModelParameters    : structure that contains  the parameters of the Hull&Whi
/// ZCMarketData : structure that contains the Zero Coupon Bond prices of the ma

/// Computation of the payoff of the ZC bond at the final time of the tree (ie t
static void ZCBond_InitialPayoffBK1D(TreeShortRate *Meth, PnlVect *ZCbondPriceVe
{
  int jminprev, jmaxprev;

  jminprev = pnl_vect_int_get(Meth->Jminimum, Meth->Ngrid);  // jmin(Ngrid)
  jmaxprev = pnl_vect_int_get(Meth->Jmaximum, Meth->Ngrid);  // jmax(Ngrid)

  pnl_vect_resize(ZCbondPriceVect, jmaxprev - jminprev + 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_InitialPayoffBK1D(PnlVect *ZCbondPriceVect, PnlVect *Option
{
  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
    }
}

/// Backward computation of the price of an option on a Zero Coupon Bond
void ZCOption_BackwardIterationBK1D(TreeShortRate *Meth, ModelParameters *ModelP
{
  double a , sigma;

  int jmin; // jmin[i+1], jmax[i+1]
  int jminprev, jmaxprev; // jmin[i], jmax [i]
  int i, j, k; // i = represents the time index. j, k represents the nodes index

  double eta_over_delta_x;
  double delta_x1, delta_x2; // delta_x1 = space step of the process x at time i
  double delta_t1, delta_t2; // time step
  double beta_x;              // quantity used in the compuation of the probabil
  double ZCPrice; //ZC price

  double current_rate;

  double Pup, Pmiddle, Pdown;

  ///*******************Parameters of the processes r, u and y ****************
  a = ModelParam->MeanReversion;
  sigma = ModelParam->RateVolatility;

  jminprev = pnl_vect_int_get(Meth->Jminimum, index_last);  // jmin(index_last)
  jmaxprev = pnl_vect_int_get(Meth->Jmaximum, index_last);  // jmax(index_last)
```

```
///** Backward computation of the option price from "index_last-1" to "index_f
for (i = index_last - 1; i >= index_first; i--)
  {
    jmin = jminprev; // jmin := jmin(i+1)

    jminprev = pnl_vect_int_get(Meth->Jminimum, i); // jminprev := jmin(i)
    jmaxprev = pnl_vect_int_get(Meth->Jmaximum, i); // jmaxprev := jmax(i)

    pnl_vect_resize(OptionPriceVect1, jmaxprev - jminprev + 1); // OptionPrice

    if (Eur_Or_Am != 0)
      {
        pnl_vect_resize(ZCbondPriceVect1, jmaxprev - jminprev + 1); // OptionP
      }

    delta_t1 = GET(Meth->t, i) - GET(Meth->t, MAX(i - 1, 0)); // Pas de temps
    delta_t2 = GET(Meth->t, i + 1) - GET(Meth->t, i); // Pas de temps entre t[

    delta_x1 = SpaceStep(delta_t1, a, sigma);  // SpaceStep (i)
    delta_x2 = SpaceStep(delta_t2, a, sigma);  // SpaceStep (i+1)

    beta_x = (delta_x1 / delta_x2) * exp(-a * delta_t2);

    // Boucle sur les noeuds
    for (j = jminprev ; j <= jmaxprev ; j++)
      {
        k = pnl_iround(j * beta_x); // index of the middle node emanating from
        eta_over_delta_x = j * beta_x - k; // quantity used in the compuation

        Pup = ProbaUp(eta_over_delta_x); // Probability of an up move from (i,
        Pmiddle = ProbaMiddle(eta_over_delta_x); // Probability of a middle mo
        Pdown = 1 - Pup - Pmiddle; // Probability of a down move from (i,j)

        current_rate = func_model_bk1d(j * delta_x1 + GET(Meth->alpha, i)); //

        LET(OptionPriceVect1, j - jminprev) = exp(-current_rate * delta_t2) *

        if (Eur_Or_Am != 0)
          {
            LET(ZCbondPriceVect1, j - jminprev) = exp(-current_rate * delta_t2
```

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

        }

      // Copy OptionPrice1 in OptionPrice2
      pnl_vect_clone(OptionPriceVect2, OptionPriceVect1);

      if (Eur_Or_Am != 0)
        {
          pnl_vect_clone(ZCbondPriceVect2, ZCbondPriceVect1);
        }

    } // END of the loop on i
}

/// Prix at time s of an option, maturing at T, on a ZC, with maturity S, using
double tr_bk1d_zcoption(TreeShortRate *Meth, ModelParameters *ModelParam, ZCMark
{
  int  i_T;
  double OptionPrice;

  PnlVect *OptionPriceVect1; // Vector of prices of the option at time i
  PnlVect *OptionPriceVect2; // Vector of prices of the option at time 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

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


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

```
  ZCBond_InitialPayoffBK1D(Meth, ZCbondPriceVect2);

  ZCOption_BackwardIterationBK1D(Meth, ModelParam, ZCbondPriceVect1, ZCbondPrice

  ZCOption_InitialPayoffBK1D(ZCbondPriceVect2, OptionPriceVect2, p);

  ///****************** Backward computation of the option price until initial t
  ZCOption_BackwardIterationBK1D(Meth, ModelParam, ZCbondPriceVect1, ZCbondPrice

  OptionPrice = GET(OptionPriceVect1, 0);

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

  return OptionPrice;

}// FIN de la fonction ZCOption


static int tr_zbo1d(int flat_flag, double r0, char *curve, double a, double sigm
{
  TreeShortRate Tr;
  ModelParameters 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.RateVolatility = sigma;

  SetTimeGrid_Tenor(&Tr, N_steps, 0, S, S);

  SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_model_bk1d, &func_model_d

  //Price of an option on a ZC
  *price = tr_bk1d_zcoption(&Tr, &ModelParams, &ZCMarket, T, S, p, r0, am);


  DeleteTreeShortRate(&Tr);
  DeleteZCMarketData(&ZCMarket);

  return OK;
}


///********************************************** PREMIA FUNCTIONS ***********
int CALC(TR_ZBOBK1D)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;


  return tr_zbo1d(ptMod->flat_flag.Val.V_INT,
                  MOD(GetYield)(ptMod),
                  MOD(GetCurve)(ptMod),
                  ptMod->a.Val.V_DOUBLE,
                  ptMod->Sigma.Val.V_PDOUBLE,
                  ptOpt->BMaturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                  ptOpt->OMaturity.Val.V_DATE - ptMod->T.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_ZBOBK1D)(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_ZBOBK1D) =
{
  "TR_BlackKarasinski1d_ZBO",
  { {"TimeStepNumber", LONG, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(TR_ZBOBK1D),
  {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(TR_ZBOBK1D),
  CHK_ok,
  MET(Init)
} ;
```