```
   Help
#include <stdlib.h>
#include "blackkarasinski1d_stdi.h"
#include "pnl/pnl_vector.h"
#include "math/InterestRateModelTree/TreeShortRate/TreeSho
    rtRate.h"
#include "math/read_market_zc/InitialYieldCurve.h"

//The "#else" part of the code will be freely available aft
    er the (year of creation of this file + 2)
#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 TreeShortRate.h)
/// ModelParameters    : structure that contains  the para
    meters of the Hull&White one factor model (see TreeShortRate.
    h)
/// ZCMarketData : structure that contains the Zero Coupon
    Bond prices of the market, or given by a constant yield-to-
    maturity (see InitialYieldCurve.h)

/// Computation of the payoff of the ZC bond at the final
    time of the tree (ie the bond maturity)
static void ZCBond_InitialPayoffBK1D(TreeShortRate* Meth,
    PnlVect* ZCbondPriceVect)
{
    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 tre
    e (ie the option maturity)
static void ZCOption_InitialPayoffBK1D(PnlVect* ZCbondPric
    eVect, PnlVect* OptionPriceVect, NumFunc_1 *p)
{
    int j;

    double ZCPrice;

    pnl_vect_resize(OptionPriceVect, ZCbondPriceVect->size)
    ;

    ///** Calcul du vecteur des payoffs a l'instant de matu
    rite de l'option

    for( j = 0 ; j<ZCbondPriceVect->size ; j++)
    {
        ZCPrice = GET(ZCbondPriceVect, j);

        LET(OptionPriceVect, j) = (p->Compute)(p->Par, ZCPr
    ice); // Payoff of the option
    }
}

/// Backward computation of the price of an option on a Ze
    ro Coupon Bond
void ZCOption_BackwardIterationBK1D(TreeShortRate* Meth,
    ModelParameters* ModelParam, PnlVect* ZCbondPriceVect1, PnlVec
    t* ZCbondPriceVect2,  PnlVect* OptionPriceVect1, PnlVect*
    OptionPriceVect2, int index_last, int index_first, NumFunc_1 *p,
     int Eur_Or_Am)
{
```

```
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 rep
resents the nodes index

double eta_over_delta_x;
double delta_x1, delta_x2; // delta_x1 = space step of
the process x at time i ; delta_x2 same at time i+1.
double delta_t1, delta_t2; // time step
double beta_x;              // quantity used in the
compuation of the probabilities. it depends only on i.
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_first", knowing those at "index_
last"**///
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);  // OptionPrice1 := Prix a l'instant i,

    if(Eur_Or_Am != 0)
    {
        pnl_vect_resize(ZCbondPriceVect1, jmaxprev-jm
inprev +1);  // OptionPrice1 := Prix a l'instant i,
    }

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

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

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

    // Boucle sur les noeuds
    for(j = jminprev ; j<= jmaxprev ; j++)
    {
        k= intapprox(j * beta_x); // index of the midd
le node emanating from (i,j)
        eta_over_delta_x = j * beta_x - k; // quantity
used in the compuation of the probabilities Pup, Pmiddle an
d Pdown.

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

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

        LET(OptionPriceVect1,j-jminprev) = exp(-
```

```
    current_rate*delta_t2) * ( Pup * GET(OptionPriceVect2, k+1-jmin)
    + Pmiddle * GET(OptionPriceVect2, k-jmin) + Pdown * GET(
    OptionPriceVect2, k-1-jmin));

            if(Eur_Or_Am != 0)
            {
                LET(ZCbondPriceVect1,j-jminprev) = exp(-
    current_rate*delta_t2) * ( Pup * GET(ZCbondPriceVect2, k+1-jmin)
    + Pmiddle * GET(ZCbondPriceVect2, k-jmin) + Pdown * GET(    ZCbondPriceVect2

                ZCPrice = GET(ZCbondPriceVect1,j-jminprev);
     // ZC price P(ti, S, r_ti=current rate)
                // In the case of american option, decide
    wether to exerice the option or not
                if( GET(OptionPriceVect1, j-jminprev) < (p-
    >Compute)(p->Par, ZCPrice))
                {
                    LET(OptionPriceVect1, j-jminprev) = (p-
    >Compute)(p->Par, ZCPrice);
                }
            }

        }

      // 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 a trinomial tree.
double tr_bk1d_zcoption(TreeShortRate* Meth, ModelParamet
    ers* ModelParam, ZCMarketData* ZCMarket, double T, double S,
     NumFunc_1 *p, double r, int Eur_Or_Am)
```

```
{
    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 payo
    ff at the maturity of the option ********************///
    i_T = IndexTime(Meth, T); // Localisation of s on the
    tree

    ZCBond_InitialPayoffBK1D(Meth, ZCbondPriceVect2);

    ZCOption_BackwardIterationBK1D(Meth, ModelParam,     ZCbondPriceVect1, ZCbon
    eVect2, Meth->Ngrid, i_T, p, 0);

    ZCOption_InitialPayoffBK1D(ZCbondPriceVect2, OptionPric
    eVect2, p);

    ///****************** Backward computation of the
    option price until initial time s ********************///
    ZCOption_BackwardIterationBK1D(Meth, ModelParam,     ZCbondPriceVect1, ZCbon
    eVect2, i_T, 0, p, Eur_Or_Am);

    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,double a,
    double sigma, double S,double T, NumFunc_1 *p,int am,int N_step
    s,double *price)
{
  TreeShortRate Tr;
  ModelParameters ModelParams;
  ZCMarketData ZCMarket;

  /* Flag to decide to read or not ZC bond datas in "initia
    lyields.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(T > GET(ZCMarket.tm,ZCMarket.Nvalue-1))
      {
          printf("{nError : time bigger than the last time
    value entered in initialyield.dat{n");
          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_der_bk1d, &func_model_inv_bk1d);

  //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),
                  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(((Option*)Opt)->Name,"ZeroCouponCallBond
    Amer")==0) || (strcmp(((Option*)Opt)->Name,"ZeroCouponPutBo
    ndEuro")==0) || (strcmp(((Option*)Opt)->Name,"ZeroCouponPut
    BondAmer")==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=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)
} ;
```

# References