```
   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"


#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_CapFloorBK1D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_CapFloorBK1D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else

// Payoff resulting from the last rate setting
static void CapFloor_InitialPayoffBK1D(TreeShortRate* Meth,
     ModelParameters* ModelParam, PnlVect* ZCbondPriceVect, Pn
    lVect* OptionPriceVect, int i_T,  NumFunc_1 *p, double pe
    riodicity, double CapFloorFixedRate)
{
    int jminprev, jmaxprev;
    int j;

    double ZCPrice;

    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

    BackwardIteration(Meth, ModelParam, OptionPriceVect,     ZCbondPriceVect, Me

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

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

        LET(OptionPriceVect, j) = (p->Compute)(p->Par, (1+
    periodicity*CapFloorFixedRate)*ZCPrice);
    }
}


// Dynamic programming + adding payoff at each reseting date
static void CapFloor_BackwardIterationBK1D(TreeShortRate*
    Meth, ModelParameters* ModelParam, NumFunc_1 *p, PnlVect*     ZCbondPriceVec
    eVect1, PnlVect* OptionPriceVect2, int index_last, int ind
    ex_first, double periodicity, double CapFloorFixedRate)
{
    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_y1 = space step of
    the process y at time i ; delta_y2 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 current_rate;
    double ZCPrice;

    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)

pnl_vect_resize(ZCbondPriceVect2, OptionPriceVect2->si
ze);

pnl_vect_set_double(ZCbondPriceVect2, 1.0); // Payoff =
 1 for a ZC bond

///** 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,
    pnl_vect_resize(ZCbondPriceVect1, jmaxprev-jminprev
 +1);

    delta_t1 = GET(Meth->t, i) - GET(Meth->t,MAX(i-1,0)
);
    delta_t2 = GET(Meth->t, i+1) - GET(Meth->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));

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

    }

  // Copy OptionPrice1 in OptionPrice2
  pnl_vect_clone(OptionPriceVect2, OptionPriceVect1);
  pnl_vect_clone(ZCbondPriceVect2, ZCbondPriceVect1);

} // END of the loop on i

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

```
    for( j = 0 ; j<ZCbondPriceVect2->size ; j++)
    {
        ZCPrice = GET(ZCbondPriceVect2, j);

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

}

// Price of a Cap/Floor using a trinomial tree
static double tr_bk1d_capfloor(TreeShortRate* Meth, ModelP
    arameters* ModelParam, ZCMarketData* ZCMarket, int NumberO
    fTimeStep, NumFunc_1 *p, double r, double periodicity,
    double first_reset_date,double contract_maturity, double CapF
    loorFixedRate)
{
    double OptionPrice, Ti2, Ti1;
    int  i, i_Ti2, i_Ti1, n;

    PnlVect* OptionPriceVect1; // Vector of prices of the
    option at i
    PnlVect* OptionPriceVect2; // Vector of prices of the
    option at i+1
    PnlVect* ZCbondPriceVect1; // Vector of prices of the
    option at i+1
    PnlVect* ZCbondPriceVect2; // Vector of prices of the
    option at i+1
    OptionPriceVect1 = pnl_vect_create(1);
    OptionPriceVect2 = pnl_vect_create(1);
    ZCbondPriceVect1 = pnl_vect_create(1);
    ZCbondPriceVect2 = pnl_vect_create(1);


    ///*************** PAYOFF at the MATURITY of the
    OPTION : T(n-1)***************///
    Ti2 = contract_maturity;
    Ti1 = Ti2 - periodicity;
    i_Ti1 = IndexTime(Meth, Ti1);
```

```
    CapFloor_InitialPayoffBK1D(Meth, ModelParam, ZCbondPric
    eVect2, OptionPriceVect2, i_Ti1,  p, periodicity, CapFloor
    FixedRate);

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

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

    for(i = n-2; i>=0; i--)
    {
        Ti1 = first_reset_date + i * periodicity; // Ti1 =
    T(i)
        Ti2 = Ti1 + periodicity;                  // Ti2 =
    T(i+1)

        i_Ti2 = IndexTime(Meth, Ti2);
        i_Ti1 = IndexTime(Meth, Ti1);

        CapFloor_BackwardIterationBK1D(Meth, ModelParam,p,     ZCbondPriceVect1,
    eVect2,  i_Ti2, i_Ti1, periodicity, CapFloorFixedRate);
    }

    ///****************** Price of the option at initial
    time s ******************///
    BackwardIteration(Meth, ModelParam, OptionPriceVect1,
    OptionPriceVect2, i_Ti1, 0, &func_model_bk1d);

    OptionPrice = GET(OptionPriceVect1, 0);

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

    return OptionPrice;

}
```

```
static int tr_capfloor1d(int flat_flag, double r0,double a,
    double sigma, double contract_maturity, double first_reset_date,
     double periodicity,double Nominal, double CapFloorFixedRa
    te, NumFunc_1 *p, long N_steps, double *price)
{
    TreeShortRate Tr;
    ModelParameters ModelParams;
    ZCMarketData ZCMarket;

    /* Flag to decide to read or not ZC bond data 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.MeanReversion  = a;
    ModelParams.RateVolatility = sigma;

    SetTimeGrid_Tenor(&Tr, N_steps, first_reset_date, contr
    act_maturity, periodicity);

    SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_
    model_bk1d, &func_model_der_bk1d, &func_model_inv_bk1d);
```

```
    *price =  Nominal * tr_bk1d_capfloor(&Tr, &ModelParams,
     &ZCMarket, N_steps, p, r0, periodicity, first_reset_date,
     contract_maturity, CapFloorFixedRate);

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

    return OK;
}



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

int CALC(TR_CapFloorBK1D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return  tr_capfloor1d(    ptMod->flat_flag.Val.V_INT,
                            MOD(GetYield)(ptMod),
                            ptMod->a.Val.V_DOUBLE,
                            ptMod->Sigma.Val.V_PDOUBLE,
                            ptOpt->BMaturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                            ptOpt->FirstResetDate.Val.V_DA
    TE-ptMod->T.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_CapFloorBK1D)(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_CapFloorBK1D)=
{
  "TR_BlackKarasinski1d_CapFloor",
  {{"TimeStepNumber per Period",LONG,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(TR_CapFloorBK1D),
  {{"Price",DOUBLE,{100},FORBID}/*,{"Delta",DOUBLE,{100},FO
    RBID} */,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_CapFloorBK1D),
  CHK_ok,
  MET(Init)
} ;
```

References