

[Help](#)

```
#include <stdlib.h>
#include "
href../../mod/blackkarasinski1d/blackkarasinski1d_std/
#include "pnl/pnl_vector.h"
#include "
href../../common/math/InterestRateModelTree/TreeShortRate/TreeShortRate_h_src
#include "
href../../common/math/read_market_zc/InitialYieldCurve_h_src.pdfmath/read_mar

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(TR_SwaptionBK1D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_SwaptionBK1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

// Payoff of the swaption at the first reseting date
static void Swaption_InitialPayoffBK1D(TreeShortRate *Meth, ModelParameters *Mod)
{
    double Ti2, Ti1;
    int i, j, i_Ti2, i_Ti1, n;
    int jminprev, jmaxprev;

    Ti2 = contract_maturity;
    Ti1 = Ti2 - periodicity;
    i_Ti1 = IndexTime(Meth, Ti1);

    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(ZCbondPriceVect2, jmaxprev - jminprev + 1);
    pnl_vect_set_double(ZCbondPriceVect2, 1 + SwaptionFixedRate * periodicity);
```

```

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

Ti1 = contract_maturity - periodicity; // Ti1 = T(n-1)
i_Ti1 = IndexTime(Meth, Ti1);

BackwardIteration(Meth, ModelParam, ZCbondPriceVect1, ZCbondPriceVect2, Meth->

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);

    pnl_vect_plus_double(ZCbondPriceVect2, SwaptionFixedRate * periodicity);

    BackwardIteration(Meth, ModelParam, ZCbondPriceVect1, ZCbondPriceVect2, i_
}

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

for (j = 0 ; j < ZCbondPriceVect2->size ; j++)
{
    LET(ZCbondPriceVect2, j) = (p->Compute)(p->Par, GET(ZCbondPriceVect1, j));
}
}

// Price of a Swaption using a trinomial tree
static double tr_bk1d_swaption(TreeShortRate *Meth, ModelParameters *ModelParam,
{
    double OptionPrice, Ti1;
    int i_Ti1;

    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(0)*****

Swaption_InitialPayoffBK1D(Meth, ModelParam, ZCbondPriceVect1, OptionPriceVect1);

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

Ti1 = first_reset_date; // Ti1 = T(0)
i_Ti1 = IndexTime(Meth, Ti1);

BackwardIteration(Meth, ModelParam, OptionPriceVect1, OptionPriceVect2, i_Ti1,

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_swaption1d(int flat_flag, double r0, char *curve, double a, double
{
    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 (contract_maturity > 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, first_reset_date, contract_maturity, periodicity);
SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_model_bk1d, &func_model_d);

*price = Nominal * tr_bk1d_swaption(&Tr, &ModelParams, &ZCMarket, N_steps, p,

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

return OK;
}

```

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

```

int CALC(TR_SwaptionBK1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return tr_swaption1d(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->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_INT,
        &(Met->Res[0].Val.V_DOUBLE));
    }
static int CHK_OPT(TR_SwaptionBK1D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PayerSwaption") == 0) || (strcmp(((Option
        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_SwaptionBK1D) =
{
    "TR_BlackKarasinski1d_Swaption",
    { {"TimeStepNumber per Period", INT, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_SwaptionBK1D),
    {{"Price", DOUBLE, {100}, FORBID} , {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(TR_SwaptionBK1D),
    CHK_ok,
    MET(Init)
} ;

```