

## Help

```
#include <stdlib.h>
#include "sg1d_std.h"
#include "pnl/pnl_vector.h"
#include "QuadraticModel.h"
#include "math/InterestRateModelTree/TreeShortRate/TreeShortRate.h"
#include "math/read_market_zc/InitialYieldCurve.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
int CALC(TR_SwaptionSG1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
static int CHK_OPT(TR_SwaptionSG1D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
#else

static void Swaption_InitialPayoffSG1D(TreeShortRate *Meth, ModelParameters *Mod)
{
    int jminprev, jmaxprev, i, j, NumberOfPayments;

    double a , sigma;
    double delta_x1; // delta_x1 = space step of the process x at time i
    double delta_t1; // time step

    double Ti, ZCPrice_T_Ti, x0, r0, current_rate, x;

    Data data1, data2;
    Omega om;

    initial_short_rate(ZCMarket, &r0, &x0);
    ZCPrice_T_Ti = 0.0;

    a = ModelParam->MeanReversion;
    sigma = ModelParam->RateVolatility;

    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(OptionPriceVect2, jmaxprev - jminprev + 1);
pnl_vect_set_double(OptionPriceVect2, 0.0);

delta_t1 = GET(Meth->t, Meth->Ngrid) - GET(Meth->t, Meth->Ngrid - 1); // Pas d
delta_x1 = SpaceStep(delta_t1, a, sigma); // SpaceStep(Ngrid)

NumberOfPayments = (int) floor((contract_maturity - option_maturity) / periodi

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

/* coefficients of P(0,T) */
bond_coeffs(ZCMarket, &data1, option_maturity, a, sigma, x0);

for (i = 1; i < NumberOfPayments; i++)
{
    Ti = option_maturity + i * periodicity;

    /* coefficients of P(0,S) */
    bond_coeffs(ZCMarket, &data2, Ti, a, sigma, x0);
    /* omega distribution of P(T,S) */
    transport(&om, data1, data2, a, sigma, x0);

    for (j = jminprev ; j <= jmaxprev ; j++)
    {
        x = j * delta_x1 + GET(Meth->alpha, Meth->Ngrid);
        current_rate = func_model_sg1d(x); // rate(Ngrid, j )

        ZCPrice_T_Ti = exp(-(om.B * current_rate + om.b * x + om.c)); // P(T,

        LET(OptionPriceVect2, j - jminprev) += periodicity * SwaptionFixedRate
    }
}

// Last payment date
Ti = contract_maturity;

/* coefficients of P(0,S) */
bond_coeffs(ZCMarket, &data2, Ti, a, sigma, x0);
/* omega distribution of P(T,S) */

```

```

transport(&om, data1, data2, a, sigma, x0);

for (j = jminprev ; j < jmaxprev ; j++)
{
    x = j * delta_x1 + GET(Meth->alpha, Meth->Ngrid);
    current_rate = func_model_sg1d(x); // rate(Ngrid, j )

    ZCPrice_T_Ti = exp(-(om.B * current_rate + om.b * x + om.c));

    LET(OptionPriceVect2, j - jminprev) += (1 + periodicity * SwaptionFixedRate);

    LET(OptionPriceVect2, j - jminprev) = ((p->Compute)(p->Par, GET(OptionPriceVect2, j - jminprev)));
}

}

/// Price of a swaption using a trinomial tree
double tr_sg1d_swaption(TreeShortRate *Meth, ModelParameters *ModelParam, ZCMarket *ZCMarket)
{
    int index_last, index_first;
    double OptionPrice;

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

    ///***** Computation of the vector of payoff at the maturity of the swaption
    Swaption_InitialPayoffSG1D(Meth, ModelParam, ZCMarket, OptionPriceVect2, p, pe);

    ///***** Backward computation of the option price until initial time
    index_last = Meth->Ngrid;
    index_first = 0;

    BackwardIteration(Meth, ModelParam, OptionPriceVect1, OptionPriceVect2, index_last, index_first);

    OptionPrice = GET(OptionPriceVect1, 0);

    pnl_vect_free(& OptionPriceVect1);
}

```

```

    pnl_vect_free(& OptionPriceVect2);

    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(&Tr, N_steps, first_reset_date);

    SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_model_sg1d, &func_model_d

    *price = Nominal * tr_sg1d_swaption(&Tr, &ModelParams, &ZCMarket, N_steps, p,

```

```

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

return OK;
}

```

```

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

```

```

int CALC(TR_SwaptionSG1D)(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_LONG,
                        &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(TR_SwaptionSG1D)(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->HelpFilenameHint = "tr_quadratic1d_swaption";
    Met->Par[0].Val.V_LONG = 500;
  }

  return OK;
}

PricingMethod MET(TR_SwaptionSG1D) =
{
  "TR_SquareGaussian1d_Swaption",
  { {"TimeStepNumber", LONG, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(TR_SwaptionSG1D),
  {"Price", DOUBLE, {100}, FORBID} , {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(TR_SwaptionSG1D),
  CHK_ok,
  MET(Init)
} ;
```