```
   Help
#include <stdlib.h>
#include "sg1d_stdi.h"
#include "pnl/pnl_vector.h"
#include "Quadraticmodel.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)
int CALC(TR_SwaptionSG1D)(void *Opt,void *Mod,Pricing
    Method *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* ModelParam, ZCMarketData* ZCMarket, PnlV
    ect* OptionPriceVect2, NumFunc_1 *p, double periodicity,
    double option_maturity,double contract_maturity, double Swaptio
    nFixedRate)
{
  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 de temps entre t[Ngrid-1] et t[Ngrid]
delta_x1 = SpaceStep(delta_t1, a, sigma);   // SpaceStep(
  Ngrid)

NumberOfPayments = (int) floor((contract_maturity-option_
  maturity )/periodicity + 0.2);

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, Ti) = P(option_maturity, Ti)

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

  // Last payement 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) * ZCPrice_T_Ti;

      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, ModelParamet
    ers* ModelParam, ZCMarketData* ZCMarket,int NumberOfTimeStep
    , NumFunc_1 *p, double r, double periodicity,double
```

```
    option_maturity,double contract_maturity, double SwaptionFixedRa
    te)
{
  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 payo
    ff at the maturity of the option ********************///
  Swaption_InitialPayoffSG1D(Meth, ModelParam, ZCMarket,
    OptionPriceVect2, p, periodicity, option_maturity, contract_matu
    rity, SwaptionFixedRate);

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

  BackwardIteration(Meth, ModelParam, OptionPriceVect1,
    OptionPriceVect2, index_last, index_first, &func_model_sg1d);

  OptionPrice = GET(OptionPriceVect1, 0);

  pnl_vect_free(& OptionPriceVect1);
  pnl_vect_free(& OptionPriceVect2);

  return OptionPrice;

}

static int tr_swaption1d(int flat_flag,double r0,double a,
    double sigma, double contract_maturity, double first_reset_date,
     double periodicity,double Nominal, double SwaptionFixedRa
    te, NumFunc_1 *p, long N_steps, 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(contract_maturity > GET(ZCMarket.tm,ZCMarket.Nvalu
    e-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(&Tr, N_steps, first_reset_date);

  SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_
    model_sg1d, &func_model_der_sg1d, &func_model_inv_sg1d);

  *price =  Nominal * tr_sg1d_swaption(&Tr, &ModelParams, &
    ZCMarket, N_steps, p, r0, periodicity, first_reset_date,
    contract_maturity, SwaptionFixedRate);

  DeleteTreeShortRate(&Tr);
```

```
  DeleteZCMarketData(&ZCMarket);

  return OK;
}



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

int CALC(TR_SwaptionSG1D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return  tr_swaption1d(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->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*)Opt)->Name,"ReceiverSwaption")==0))
    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)
} ;
```

References