```
   Help
#include <stdlib.h>
#include "sg1d_stdi.h"
#include "pnl/pnl_mathtools.h"
#include "math/read_market_zc/InitialYieldCurve.h"
#include "Quadraticmodel.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)
int CALC(CF_PayerSwaptionSG1D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
static int CHK_OPT(CF_PayerSwaptionSG1D)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
#else

///* Computation the function phi used to find the Criti
    cal Rate in the Jamishidian decomposition
static double phi(ZCMarketData* ZCMarket, double r, double
    periodicity, double option_maturity, double contract_matu
    rity, double SwaptionFixedRate, double a, double sigma)
{
    int i, nb_payement;
    double ci, sum,sum_der,ti;

    double ZCPrice_T_ti;
    double r0, x0, x;

    Data data1, data2;
    Omega om;

    ZCPrice_T_ti = 0.;
    r0=0.0; x0=0.0;
```

```
initial_short_rate(ZCMarket, &r0, &x0);

sum=0.;
sum_der=0.;

ci = periodicity * SwaptionFixedRate;
ti = option_maturity;

nb_payement = (int)((contract_maturity-option_maturity)
/periodicity);

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

x = sqrt(r);

for(i=1; i<=nb_payement; i++)
{
    ti += 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);

    ZCPrice_T_ti = exp(-(om.B*r + om.b*M_SQRT2*x + om.
c));

    sum += ci * ZCPrice_T_ti;

    sum_der -= ci * ZCPrice_T_ti * (om.B + om.b/x);
}

sum += ZCPrice_T_ti;

sum_der -= ZCPrice_T_ti * (om.B + om.b/x);

return (sum-1.)/sum_der;
```

```
}


/// Computation of Critical Rate in the Jamishidian decompo
    sition, with the newton method to find zero of a function
static double Critical_Rate(ZCMarketData* ZCMarket, double
    r_initial, double periodicity, double option_maturity,
    double contract_maturity, double SwaptionFixedRate, double a,
    double sigma)
{
  double previous,current;
  int nbr_iterations;

  const double precision = 0.000001;

  current = r_initial;
  nbr_iterations = 0;

  do
    {
      nbr_iterations++;
      previous =current;
      current=current-phi(ZCMarket, current, periodicity,
    option_maturity, contract_maturity, SwaptionFixedRate, a, sigma)
    ;
    } while((fabs(previous-current) > precision) && (nbr_
    iterations <= 50));

  return current;
}

///* Payer Swaption price as a combination of ZC Put
    option prices
static int cf_ps1d(int flat_flag, double r_t, double Nomina
    l, double periodicity,
                   double option_maturity, double contract_
    maturity,
                   double SwaptionFixedRate, double a,
    double sigma,double *price)
{
    int i, nb_payement;
```

```
double ci, sum ,ti;

double x0, critical_r, Strike_i, PutOptionPrice;
Data data1, data2;
Omega om;
ZCMarketData ZCMarket;

PutOptionPrice = 0.; // to avoid warning

// Flag to decide to read or not ZC bond datas 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 = r_t;
}

else
{
    ZCMarket.FlatOrMarket = 1;
    ReadMarketData(&ZCMarket);
    r_t = -log(BondPrice(INC, &ZCMarket))/INC;

    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);
    }
}

x0 = sqrt(2* r_t);

bond_coeffs(&ZCMarket, &data1, option_maturity, a, si
gma, x0);

ti = option_maturity;
ci = periodicity * SwaptionFixedRate;
```

```
    nb_payement = (int)((contract_maturity-option_maturity)
    /periodicity);

    critical_r = Critical_Rate(&ZCMarket, r_t, periodicity,
     option_maturity, contract_maturity, SwaptionFixedRate, a,
     sigma);

    sum=0.;

    for(i=1; i<=nb_payement; i++)
    {
        ti += 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);

        Strike_i = exp(-(om.B*critical_r + om.b*sqrt(2*cr
    itical_r) + om.c));

        PutOptionPrice = zb_put_quad1d(&ZCMarket, a, sigma,
     option_maturity, ti, Strike_i);

        sum += ci * PutOptionPrice;
    }

    sum += PutOptionPrice;

    *price = Nominal * sum;

    DeleteZCMarketData(&ZCMarket);

    return OK;
}

int CALC(CF_PayerSwaptionSG1D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
```

```
  TYPEMOD* ptMod=(TYPEMOD*)Mod;


  return cf_ps1d( ptMod->flat_flag.Val.V_INT,
                  MOD(GetYield)(ptMod),
                  ptOpt->Nominal.Val.V_PDOUBLE,
                  ptOpt->ResetPeriod.Val.V_DATE,
                  ptOpt->OMaturity.Val.V_DATE-ptMod->T.Val.
    V_DATE,
                  ptOpt->BMaturity.Val.V_DATE-ptMod->T.Val.
    V_DATE,
                  ptOpt->FixedRate.Val.V_PDOUBLE,
                  ptMod->a.Val.V_DOUBLE,
                  ptMod->Sigma.Val.V_PDOUBLE,
                  &(Met->Res[0].Val.V_DOUBLE));
}
static int CHK_OPT(CF_PayerSwaptionSG1D)(void *Opt, void *
    Mod)
{
  return strcmp( ((Option*)Opt)->Name,"PayerSwaption");
}
#endif //PremiaCurrentVersion


static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;
       Met->HelpFilenameHint = "    cf_quadratic1d_payerswaption";
    }

  return OK;
}

PricingMethod MET(CF_PayerSwaptionSG1D)=
{
  "CF_SquareGaussian1d_PayerSwaption",
  {{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(CF_PayerSwaptionSG1D),
  {{"Price",DOUBLE,{100},FORBID},{" ",PREMIA_NULLTYPE,{0},
```

```
    FORBID}},
  CHK_OPT(CF_PayerSwaptionSG1D),
  CHK_ok,
  MET(Init)
} ;
```

References