Help

```
#include "lmm1d_stdi.h"
#include "pnl/pnl_basis.h"
#include "math/mc_lmm_glassermanzhao.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+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(MC_LongstaffSchwartz_BermudanSwaption)(
    void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_LongstaffSchwartz_BermudanSwaption)(void *Opt,
    void *Mod,PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/** Price of bermudan swaption using Longstaff-Schwartz alg
    orithm
 * @param LS_Price price by Longstaff-Schwartz algorithm on
     exit.
 * @param Nominal nominal of swaption
 * @param NbrMCsimulation the number of samples
 * @param ptLib Libor structure contains initial value of
    libor rates
 * @param ptBermSwpt Swaption structure contains bermudan
    swaption information
 * @param ptVol Volatility structure contains libor
    volatility deterministic function
 * @param generator the index of the random generator to
    be used
 * @param basis_name regression basis
 * @param DimApprox dimension of regression basis
 * @param NbrStepPerTenor number of steps of discretization
     between T(i) and T(i+1)
 * @param flag_numeraire measure under wich simulation is
    done.
```

```
 * flag_numeraire=0->Terminal measure, flag_numeraire=1->
    Spot measure
 * Rmk: Libor rates are simulated using the method proposed
     by Glasserman-Zhao.
 */
static void MC_BermSwaption_LongstaffSchwartz(double *LS_
    Price, double Nominal, int NbrMCsimulation, NumFunc_1 *p,
    Libor *ptLib, Swaption *ptBermSwpt, Volatility *ptVol, int     generator, in
    flag_numeraire)
{
  int alpha, beta, m, k, N, NbrExerciseDates, time_index,
    save_brownian, save_all_paths, start_index, end_index, Nstep
    s, nbr_var_explicatives;
  double tenor, regressed_value, payoff, numeraire_0;
  double *VariablesExplicatives;

  Libor *ptL_current;
  Swaption *ptSwpt;
  PnlMat *LiborPathsMatrix, *BrownianMatrixPaths;
  PnlMat *ExplicativeVariables;
  PnlVect *OptimalPayoff;
  PnlVect *RegressionCoeffVect;
  PnlBasis *basis;

  //Nfac = ptVol->numberOfFactors;
  N = ptLib->numberOfMaturities;
  tenor = ptBermSwpt->tenor;
  alpha = (int)(ptBermSwpt->swaptionMaturity/tenor); // T(
    alpha) is the swaption maturity
  beta  = (int)(ptBermSwpt->swapMaturity/tenor); // T(beta)
     is the swap maturity
  NbrExerciseDates = beta-alpha;
  start_index = 0;
  end_index = beta-1;
  Nsteps = end_index - start_index;

  save_brownian = 0;
  save_all_paths = 1;
  nbr_var_explicatives = 2;

  VariablesExplicatives = malloc(nbr_var_explicatives*size
```

```
  of(double));
ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nb
  r_var_explicatives); // Explicatives variables
OptimalPayoff = pnl_vect_create(NbrMCsimulation);
RegressionCoeffVect = pnl_vect_new();
LiborPathsMatrix = pnl_mat_new(); // LiborPathsMatrix     contains all the tra
BrownianMatrixPaths = pnl_mat_new(); // We store also th
  e brownian values to be used a explicatives variables.

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_e
  xplicatives);

mallocLibor(&ptL_current, N, tenor, 0.1);

// ptSwpt := contains the information about the swap to
  be be exerced at each exercice date.
// The maturity of the swap stays the same.
mallocSwaption(&ptSwpt, ptBermSwpt->swaptionMaturity, pt
  BermSwpt->swapMaturity, 0.0, ptBermSwpt->strike, tenor);

numeraire_0 = Numeraire(0, ptLib, flag_numeraire);

// Simulation the "NbrMCsimulation" paths of Libor rates.
   We also store brownian motion values.
Sim_Libor_Glasserman(start_index, end_index, ptLib, pt    Vol, generator, NbrM
  paths, LiborPathsMatrix, save_brownian, BrownianMatrixPaths,
  flag_numeraire);

ptSwpt->swaptionMaturity = ptBermSwpt->swapMaturity - ten
  or; // Last exerice date.
time_index = end_index;

// At the last exercice date, price of the option = payo
  ff.
for (m=0; m<NbrMCsimulation; m++)
  {
    pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix,
   time_index + m*Nsteps);
    LET(OptimalPayoff, m) = Swaption_Payoff_Discounted(pt
  L_current, ptSwpt, p, flag_numeraire);
  }
```

```
for (k=NbrExerciseDates-1; k>=1; k--)
  {
    ptSwpt->swaptionMaturity -= tenor; // k'th exercice
  date
    time_index -=1;

    for (m=0; m<NbrMCsimulation; m++)
      {
        pnl_mat_get_row(ptL_current->libor, LiborPathsM
  atrix, time_index + m*Nsteps);
        MLET(ExplicativeVariables, m, 0) = computeSwapR
  ate(ptL_current, time_index, time_index, beta);
        MLET(ExplicativeVariables, m, 1) = GET(ptL_
  current->libor, time_index);
      }
    // Least square fitting
    pnl_basis_fit_ls(basis,RegressionCoeffVect, Explicati
  veVariables, OptimalPayoff);

    // Equation de programmation dynamique.
    for (m=0; m<NbrMCsimulation; m++)
      {
        pnl_mat_get_row(ptL_current->libor, LiborPathsM
  atrix, time_index + m*Nsteps);
        payoff = Swaption_Payoff_Discounted(ptL_current,
  ptSwpt, p, flag_numeraire);

        // If the payoff is null, the OptimalPayoff doesn
  't change.
        if (payoff>0)
          {
            VariablesExplicatives[0] = computeSwapRate(pt
  L_current, time_index, time_index, beta);
            VariablesExplicatives[1] = GET(ptL_current->
  libor, time_index);

            regressed_value = pnl_basis_eval(basis,Regres
  sionCoeffVect, VariablesExplicatives);

            if (payoff > regressed_value)
```

```
                {
                  LET(OptimalPayoff, m) = payoff;
                }
            }
        }
    }

  // The price at date 0 is the conditional expectation of
    OptimalPayoff, ie it's empirical mean.
  *LS_Price = pnl_vect_sum(OptimalPayoff)/NbrMCsimulation;

  *LS_Price *= (double) (numeraire_0 * Nominal);

  free(VariablesExplicatives);
  pnl_basis_free (&basis);
  pnl_mat_free(&LiborPathsMatrix);
  pnl_mat_free(&ExplicativeVariables);

  pnl_vect_free(&OptimalPayoff);
  pnl_vect_free(&RegressionCoeffVect);
  pnl_mat_free(&BrownianMatrixPaths);

  freeSwaption(&ptSwpt);
  freeLibor(&ptL_current);
}


static int MCLongstaffSchwartz(NumFunc_1 *p, double l0,
    double sigma_const, int nb_factors, double swap_maturity,
    double swaption_maturity,  double Nominal, double swaption_stri
    ke, double tenor, long nb_MC, int generator, int basis_na
    me, int DimApprox, int NbrStepPerTenor, int flag_numeraire,
    double *swaption_price)
{
  Volatility *ptVol;
  Libor *ptLib;
  Swaption *ptBermSwpt;
  int init_mc;
  int Nbr_Maturities;

  Nbr_Maturities = (int) (swap_maturity/tenor);
```

```
  mallocLibor(&ptLib , Nbr_Maturities, tenor,l0);
  mallocVolatility(&ptVol , nb_factors, sigma_const);
  mallocSwaption(&ptBermSwpt, swaption_maturity, swap_matu
    rity, 0.0, swaption_strike, tenor);

  init_mc = pnl_rand_init(generator, nb_factors, nb_MC);
  if (init_mc != OK) return init_mc;

  MC_BermSwaption_LongstaffSchwartz(swaption_price, Nomina
    l, nb_MC, p, ptLib, ptBermSwpt, ptVol, generator, basis_na
    me, DimApprox, NbrStepPerTenor, flag_numeraire);

  freeLibor(&ptLib);
  freeVolatility(&ptVol);
  freeSwaption(&ptBermSwpt);

  return init_mc;
}

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

  return MCLongstaffSchwartz(    ptOpt->PayOff.Val.V_
    NUMFUNC_1,
                                 ptMod->l0.Val.V_PDOUBLE,
                                 ptMod->Sigma.Val.V_PDOUB
    LE,
                                 ptMod->NbFactors.Val.V_
    ENUM.value,
                                 ptOpt->BMaturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,
                                 ptOpt->OMaturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,
                                 ptOpt->Nominal.Val.V_PDOUB
    LE,
                                 ptOpt->FixedRate.Val.V_PDO
    UBLE,
```

```
                                           ptOpt->ResetPeriod.Val.V_
    DATE,
                                           Met->Par[0].Val.V_LONG,
                                           Met->Par[1].Val.V_ENUM.val
    ue,
                                           Met->Par[2].Val.V_ENUM.val
    ue,
                                           Met->Par[3].Val.V_INT,
                                           Met->Par[4].Val.V_INT,
                                           Met->Par[5].Val.V_ENUM.val
    ue,
                                           &(Met->Res[0].Val.V_
    DOUBLE));
}

static int CHK_OPT(MC_LongstaffSchwartz_BermudanSwaption)(
    void *Opt, void *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"PayerBermudanSwaption")
    ==0) || (strcmp(((Option*)Opt)->Name,"
    ReceiverBermudanSwaption")==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=50000;
      Met->Par[1].Val.V_ENUM.value=0;
      Met->Par[1].Val.V_ENUM.members=&PremiaEnumRNGs;
      Met->Par[2].Val.V_ENUM.value=0;
      Met->Par[2].Val.V_ENUM.members=&PremiaEnumBasis;
      Met->Par[3].Val.V_INT=10;
      Met->Par[4].Val.V_INT=1;
```

```
      Met->Par[5].Val.V_ENUM.value=0;
      Met->Par[5].Val.V_ENUM.members=&PremiaEnumAfd;

    }

  return OK;
}


PricingMethod MET(MC_LongstaffSchwartz_BermudanSwaption)=
{
  "MC_LongstaffSchwartz_BermudanSwaption",
  {
    {"N Simulation",LONG,{100},ALLOW},
    {"RandomGenerator",ENUM,{100},ALLOW},
    {"Basis",ENUM,{100},ALLOW},
    {"Dimension Approximation",INT,{100},ALLOW},
    {"Nbr discretisation step per periode",INT,{100},ALLOW}
    ,
    {"Martingale Measure",ENUM,{100},ALLOW},
    {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_LongstaffSchwartz_BermudanSwaption),
  {{"Price",DOUBLE,{100},FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_LongstaffSchwartz_BermudanSwaption),
  CHK_ok,
  MET(Init)
};
```

References