Help

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

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2010+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_Schoenmakers_BermudanSwaption)(void *
    Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_Schoenmakers_BermudanSwaption)(void *Opt,void *
    Mod,PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


/**
 * Lower bound for bermudan swaption using Longstaff-Schwa
    rtz algorithm
 * We store the regression coefficients in a matrix LS_Reg
    ressionCoeffMat
 * We also compute the coefficients regression to estimate
    the conditional expectation needed in Schoenmakers et al.
    algorithm.
 * These coefficients are stored in a matrix Sch_Regression
    CoeffMat
 * @param LS_LowerPrice lower price by Longstaff-Schwartz
    algorithm on exit
 * @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
 * @param LS_RegressionCoeffMat contains Longstaff-Schwartz
    algorithm regression coefficients
 * @param Sch_RegressionCoeffMat contains regression coeffi
   cients needed in Schoenmakers et al. algorithm.
 * Rmk: Libor rates are simulated using the method proposed
    by Glasserman-Zhao.
 */
static void MC_BermSwaption_LongstaffSchwartz(double *LS_
    LowerPrice, int NbrMCsimulation, NumFunc_1 *p, Libor *ptLib,
     Swaption *ptBermSwpt, Volatility *ptVol, int generator,
    int basis_name, int DimApprox, int NbrStepPerTenor, int flag_
    numeraire, PnlMat *LS_RegressionCoeffMat, PnlMat *Sch_Regressi
    onCoeffMat)
{
  int alpha, beta, i, j, m, k, N, NbrExerciseDates, time_i
    ndex, save_brownian, save_all_paths, start_index, end_ind
    ex, Nsteps, nbr_var_explicatives, Nfac;
  double tenor, regressed_value, payoff,dW;
  double *VariablesExplicatives;

  Libor *ptLib_current;
  Swaption *ptSwpt_current;
  PnlVect *OptimalPayoff, *LS_RegCoeffVect, *Sch_RegCoeffV
    ect, *ToRegressSch_Vect;
  PnlMat *LiborPathsMatrix, *BrownianPathsMatrix, *Explic
    ativeVariables;
  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 = 1;
save_all_paths = 1;
nbr_var_explicatives = Nfac;

VariablesExplicatives = malloc(nbr_var_explicatives*size
  of(double));
ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nb
  r_var_explicatives); // Explicatives variables
OptimalPayoff = pnl_vect_create(NbrMCsimulation);
ToRegressSch_Vect = pnl_vect_create(NbrMCsimulation);
LS_RegCoeffVect = pnl_vect_create(0);
Sch_RegCoeffVect = pnl_vect_create(0);
LiborPathsMatrix = pnl_mat_create(0, 0); // LiborPathsM
  atrix contains all the trajectories.
BrownianPathsMatrix = pnl_mat_create(0, 0); // We store
  also the brownian values to be used as explicatives variables
  .

pnl_mat_resize(LS_RegressionCoeffMat, NbrExerciseDates-1,
   DimApprox);
pnl_mat_resize(Sch_RegressionCoeffMat, (NbrExerciseDates-
  1)*Nfac, DimApprox);

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

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

// ptSwpt_current := contains the information about the
  swap to be be exerced at each exercice date.
// The maturity of the swap stays the same.
mallocSwaption(&ptSwpt_current, ptBermSwpt->swaptionMatu
```

```
  rity, ptBermSwpt->swapMaturity, 0.0, ptBermSwpt->strike, ten
  or);

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, BrownianPathsMatrix,
  flag_numeraire);

ptSwpt_current->swaptionMaturity = ptBermSwpt->swapMatu
  rity - tenor; // 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(ptLib_current->libor, LiborPathsMatr
  ix, time_index + m*Nsteps);
    LET(OptimalPayoff, m) = Swaption_Payoff_Discounted(pt
  Lib_current, ptSwpt_current, p, flag_numeraire);
  }

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

    /** Least square fitting. **/
    for (m=0; m<NbrMCsimulation; m++)
      {
        for (j=0; j<Nfac; j++)
          {
            MLET(ExplicativeVariables, m, j) = MGET(Brow
  nianPathsMatrix, time_index-1 + m*Nsteps, j);
          }
      }
```

```
  pnl_basis_fit_ls(basis,LS_RegCoeffVect, ExplicativeV
ariables, OptimalPayoff);
  pnl_mat_set_row(LS_RegressionCoeffMat, LS_RegCoeffVec
t, k-1); // Store regression coefficients

  /** Regression coefficients needed in Schoenmakers et
 al. algorithm. **/
  for (j=0; j<Nfac; j++)
    {
      for (m=0; m<NbrMCsimulation; m++)
        {
          for (i=0; i<Nfac; i++)
            {
              VariablesExplicatives[i] = MGET(BrownianP
athsMatrix, time_index-1 + m*Nsteps, i);
            }
          regressed_value = pnl_basis_eval(basis,LS_Reg
CoeffVect, VariablesExplicatives);

          dW = MGET(BrownianPathsMatrix, time_index +
m*Nsteps, j)-MGET(BrownianPathsMatrix, time_index-1 + m*Ns
teps, j);
          LET(ToRegressSch_Vect, m) = (dW/tenor) * (GET
(OptimalPayoff, m)-regressed_value);
        }

      pnl_basis_fit_ls(basis,Sch_RegCoeffVect, Explic
ativeVariables, ToRegressSch_Vect);
      pnl_mat_set_row(Sch_RegressionCoeffMat, Sch_Reg
CoeffVect, (k-1)*Nfac + j);
    }

  /** Dynamical programing. **/
  for (m=0; m<NbrMCsimulation; m++)
    {
      pnl_mat_get_row(ptLib_current->libor, LiborPathsM
atrix, time_index + m*Nsteps);
      payoff = Swaption_Payoff_Discounted(ptLib_
current, ptSwpt_current, p, flag_numeraire);

      // If the payoff is null, the OptimalPayoff doesn
```

```
    t change.
          if (payoff>0)
            {
              for (j=0; j<Nfac; j++)
                {
                  VariablesExplicatives[j] = MGET(BrownianP
    athsMatrix, time_index-1 + m*Nsteps, j);
                }

              regressed_value = pnl_basis_eval(basis,LS_Reg
    CoeffVect, VariablesExplicatives);

              if (payoff > regressed_value)
                {
                  LET(OptimalPayoff, m) = payoff;
                }
            }
        }
    }

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

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

  pnl_vect_free(&OptimalPayoff);
  pnl_vect_free(&LS_RegCoeffVect);
  pnl_vect_free(&Sch_RegCoeffVect);
  pnl_vect_free(&ToRegressSch_Vect);

  freeSwaption(&ptSwpt_current);
  freeLibor(&ptLib_current);
}
```

```
/** Upper bound for bermudan swaption using Schoenmakers et
     al. algorithm.
 * @param SwaptionPriceUpper upper bound for the price on
    exit.
 * @param NbrMCsimulationDual number of simulation in Schoe
    nmakers et al. algorithm.
 * @param NbrMCsimulationPrimal number of simulation in Lon
    gstaff-Schwartz algorithm.
 */
static void Schoenmakers(double *SwaptionPriceUpper,
    double Nominal, long NbrMCsimulationDual, long NbrMCsimulationP
    rimal, NumFunc_1 *p, Libor *ptLib, Swaption *ptBermSwpt,
    Volatility *ptVol, int generator, int basis_name, int DimApprox,
    int NbrStepPerTenor, int flag_numeraire)
{
  int i, j, m, N, k, Nfac, alpha, beta, Nsteps, save_all_
    paths, save_brownian;
  int NbrExerciseDates, start_index, end_index, nbr_var_ex
    plicatives;
  double tenor, payoff, numeraire_0, ContinuationValue, Low
    erPrice_0, LowerPrice_alpha;
  double DoobMeyerMartingale, MaxVariable, Delta_0, dW, Z;
  double *VariablesExplicatives;


  PnlMat *LiborPathsMatrix, *BrownianPathsMatrix;
  PnlMat *LS_RegressionCoeffMat, *Sch_RegressionCoeffMat;
  PnlVect *Sch_RegCoeffVect, *LS_RegressionCoeffVect;
  PnlBasis *basis;

  Libor *ptLib_current;
  Swaption *ptSwpt_current;

  Nfac = ptVol->numberOfFactors;
  N = ptLib->numberOfMaturities;
  tenor = ptBermSwpt->tenor;
  alpha = (int)(ptBermSwpt->swaptionMaturity/tenor); // T(
    alpha) is the swaption maturity. T(i) = i*tenor.
  beta  = (int)(ptBermSwpt->swapMaturity/tenor); // T(beta)
     is the swap maturity
  NbrExerciseDates = beta-alpha;
```

```
nbr_var_explicatives = Nfac;
VariablesExplicatives = malloc(nbr_var_explicatives*size
  of(double));
basis = pnl_basis_create(basis_name, DimApprox, nbr_var_e
  xplicatives);

LS_RegressionCoeffVect = pnl_vect_create(0);
LS_RegressionCoeffMat = pnl_mat_create(0, 0);
Sch_RegCoeffVect = pnl_vect_create(0);
Sch_RegressionCoeffMat = pnl_mat_create(0, 0);

LiborPathsMatrix = pnl_mat_create(0, 0);
BrownianPathsMatrix = pnl_mat_create(0, 0);

mallocLibor(&ptLib_current , N, tenor, 0.);

numeraire_0 = Numeraire(0, ptLib, flag_numeraire);

// ptSwpt_current := le swap qui sera exerce à chaque da
  te de la bermudeene. sa maturite reste fixe.
mallocSwaption(&ptSwpt_current, ptBermSwpt->swaptionMatu
  rity, ptBermSwpt->swapMaturity, 0.0, ptBermSwpt->strike, ten
  or);

// calcul de la borne inf du prix et des coefficients de
  regression.
MC_BermSwaption_LongstaffSchwartz(&LowerPrice_0, NbrMCs
  imulationPrimal, p, ptLib, ptBermSwpt, ptVol, generator,
  basis_name, DimApprox, NbrStepPerTenor/2+1, flag_numeraire,
  LS_RegressionCoeffMat, Sch_RegressionCoeffMat);

Delta_0 = 0;

save_brownian = 2;  // save_brownian = 2. We also save
  intermediate steps.
save_all_paths = 1; // If save_all_paths=1, we store the
  simulated value of libors at each date T(i).

start_index = 0;
end_index = beta-1;
```

```
Nsteps = end_index - start_index;

// Simulate "NbrMCsimulationDual" paths
Sim_Libor_Glasserman(start_index, end_index, ptLib, pt    Vol, generator, NbrM
  all_paths, LiborPathsMatrix, save_brownian, BrownianPathsM
  atrix, flag_numeraire);

for (m=0; m<NbrMCsimulationDual; m++)
  {
    start_index = alpha;

    pnl_mat_get_row(ptLib_current->libor, LiborPathsMatr
  ix, start_index + m*Nsteps);
    ptSwpt_current->swaptionMaturity = ptBermSwpt->swapt
  ionMaturity; // First exerice date.
    payoff = Swaption_Payoff_Discounted(ptLib_current, pt
  Swpt_current, p, flag_numeraire);

    pnl_mat_get_row(LS_RegressionCoeffVect, LS_Regression
  CoeffMat, 0);

    for (j=0; j<Nfac; j++)
      {
        VariablesExplicatives[j] = MGET(BrownianPathsMatr
  ix, start_index*NbrStepPerTenor-1 + m*NbrStepPerTenor*Nstep
  s, j);
      }
    ContinuationValue = pnl_basis_eval(basis,LS_Regressi
  onCoeffVect, VariablesExplicatives);
    LowerPrice_alpha = MAX(ContinuationValue, payoff); //
   Price of the option at t=T(alpha), using Longstaff/Schwa
  rtz.

    DoobMeyerMartingale = LowerPrice_alpha; // Martingal
  e value at t=T(alpha).

    MaxVariable = payoff-DoobMeyerMartingale; // Value of
   Duale Variable at t=T(alpha).

    for (k=0; k<NbrExerciseDates-1; k++)
      {
```

```
      start_index = alpha + k;
      end_index = start_index+1;

      for (i=1 ; i<=NbrStepPerTenor; i++)
        {
          for (j=0; j<Nfac; j++)
            {
              VariablesExplicatives[j] = MGET(BrownianP
athsMatrix, i-1 + start_index*NbrStepPerTenor-1 + m*NbrStep
PerTenor*Nsteps, j);
            }

          // Here we compute the stochatic integral of
Z process with respect to brownian motion W.
          for (j=0; j<Nfac; j++)
            {
              pnl_mat_get_row(Sch_RegCoeffVect, Sch_Reg
ressionCoeffMat, k*Nfac+j);
              Z = pnl_basis_eval(basis,Sch_RegCoeffVec
t, VariablesExplicatives);

              dW = MGET(BrownianPathsMatrix, i + start_
index*NbrStepPerTenor-1 + m*NbrStepPerTenor*Nsteps, j);
              dW -= VariablesExplicatives[j];
              DoobMeyerMartingale += Z * dW;
            }
        }

      ptSwpt_current->swaptionMaturity += tenor;
      pnl_mat_get_row(ptLib_current->libor, LiborPathsM
atrix, end_index + m*Nsteps);
      payoff = Swaption_Payoff_Discounted(ptLib_
current, ptSwpt_current, p, flag_numeraire);

      MaxVariable = MAX(MaxVariable, payoff-DoobMeyerM
artingale); // Value of Duale Variable.
    }

  Delta_0 += MaxVariable; // somme de MonteCarlo
}
```

```
  Delta_0 /= NbrMCsimulationDual;

  *SwaptionPriceUpper = (numeraire_0 * Nominal) * (LowerP
    rice_0 + 0.5*Delta_0);

  free(VariablesExplicatives);

  pnl_basis_free (&basis);
  pnl_mat_free(&LiborPathsMatrix);
  pnl_mat_free(&BrownianPathsMatrix);
  pnl_mat_free(&Sch_RegressionCoeffMat);
  pnl_mat_free(&LS_RegressionCoeffMat);

  pnl_vect_free(&Sch_RegCoeffVect);
  pnl_vect_free(&LS_RegressionCoeffVect);

  freeSwaption(&ptSwpt_current);
  freeLibor(&ptLib_current);
}

static int MCSchoenmakers(NumFunc_1 *p, double l0, double
    sigma_const, int nb_factors, double swap_maturity, double
    swaption_maturity,  double Nominal, double swaption_strike,
    double tenor, long NbrMCsimulationPrimal, long NbrMCsimulationD
    ual, int generator, int basis_name, int DimApprox, int Nb
    rStepPerTenor, int flag_numeraire, double *swaption_price_upp
    er)
{
  Volatility *ptVol;
  Libor *ptLib;
  Swaption *ptBermSwpt;
  int init_mc;
  int Nbr_Maturities;

  Nbr_Maturities = (int)(swap_maturity/tenor + 0.1);

  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, NbrMCsimu
    lationPrimal);
  if (init_mc != OK) return init_mc;

  Schoenmakers(swaption_price_upper, Nominal, NbrMCsimulat
    ionDual, NbrMCsimulationPrimal, p, ptLib, ptBermSwpt, pt    Vol, generator,
    numeraire);

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

  return init_mc;
}

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

  return MCSchoenmakers(  ptOpt->PayOff.Val.V_NUMFUNC_1,
                          ptMod->l0.Val.V_PDOUBLE,
                          ptMod->Sigma.Val.V_PDOUBLE,
                          ptMod->NbFactors.Val.V_ENUM.value
    ,
                          ptOpt->BMaturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                          ptOpt->OMaturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                          ptOpt->Nominal.Val.V_PDOUBLE,
                          ptOpt->FixedRate.Val.V_PDOUBLE,
                          ptOpt->ResetPeriod.Val.V_DATE,
                          Met->Par[0].Val.V_LONG,
                          Met->Par[1].Val.V_LONG,
                          Met->Par[2].Val.V_ENUM.value,
                          Met->Par[3].Val.V_ENUM.value,
                          Met->Par[4].Val.V_INT,
                          Met->Par[5].Val.V_INT,
                          Met->Par[6].Val.V_ENUM.value,
                          &(Met->Res[0].Val.V_DOUBLE));
```

```
}

static int CHK_OPT(MC_Schoenmakers_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_LONG=10000;
      Met->Par[2].Val.V_ENUM.value=0;
      Met->Par[2].Val.V_ENUM.members=&PremiaEnumMCRNGs;
      Met->Par[3].Val.V_ENUM.value=0;
      Met->Par[3].Val.V_ENUM.members=&PremiaEnumBasis;
      Met->Par[4].Val.V_INT=10;
      Met->Par[5].Val.V_INT=10;
      Met->Par[6].Val.V_ENUM.value=0;
      Met->Par[6].Val.V_ENUM.members=&PremiaEnumAfd;
    }

  return OK;
}


PricingMethod MET(MC_Schoenmakers_BermudanSwaption)=
{
  "MC_Schoenmakers_BermudanSwaption",
  {
    {"N iterations Primal",LONG,{100},ALLOW},
```

```
    {"N iterations Dual",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_Schoenmakers_BermudanSwaption),
  {{"Price",DOUBLE,{100},FORBID}, {" ",PREMIA_NULLTYPE,{0},
    FORBID}},
  CHK_OPT(MC_Schoenmakers_BermudanSwaption),
  CHK_ok,
  MET(Init)
};
```

References