Help

```
#include  "
href../../../mod/lmm1d/lmm1d_stdi/lmm1d_stdi_h_src.pdflmm1d_stdi.h"
#include "pnl/pnl_basis.h"
#include "
href../../../common/math/mc_lmm_glassermanzhao_h_src.pdfmath/mc_lmm_glassermanzh
#include "
href../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
static int CHK_OPT(MC_AndersenBroadie_BermudanSwaption)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_AndersenBroadie_BermudanSwaption)(void *Opt, void *Mod, PricingMetho
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/** Lower bound for bermudan swaption using Longstaff-Schwartz algorithm
 * We store the regression coefficients in a matrix LS_RegressionCoeffMat
 * @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 fun
 * @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
 * @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
 * Rmk: Libor rates are simulated using the method proposed by Glasserman-Zhao.
 */
static void MC_BermSwaption_LongstaffSchwartz(double *LS_LowerPrice, int NbrMCsi
{
  int alpha, beta, j, m, k, N, NbrExerciseDates, time_index, save_brownian, save
  double tenor, regressed_value, payoff;
```

```
double *VariablesExplicatives;

Libor *ptLib_current;
Swaption *ptSwpt_current;
PnlMat *LiborPathsMatrix, *BrownianMatrixPaths;
PnlMat *ExplicativeVariables;
PnlVect *OptimalPayoff;
PnlVect *LS_RegressionCoeffVect;
PnlBasis *basis;

Nfac = ptVol->numberOfFactors;
N = ptLib->numberOfMaturities;
tenor = ptBermSwpt->tenor;
alpha = (int)(ptBermSwpt->swaptionMaturity / tenor); // T(alpha) is the swapti
beta  = (int)(ptBermSwpt->swapMaturity / tenor); // T(beta) is the swap maturi
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 * sizeof(double));
ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nbr_var_explicatives);
OptimalPayoff = pnl_vect_create(NbrMCsimulation);
LS_RegressionCoeffVect = pnl_vect_create(0);
LiborPathsMatrix = pnl_mat_create(0, 0); // LiborPathsMatrix contains all the
BrownianMatrixPaths = pnl_mat_create(0, 0); // We store also the brownian valu

pnl_mat_resize(LS_RegressionCoeffMat, NbrExerciseDates - 1, DimApprox);

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives);

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

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

```
//numeraire_0 = Numeraire(0, ptLib, flag_numeraire);

// Simulation the "NbrMCsimulation" paths of Libor rates. We also store browni
Sim_Libor_Glasserman(start_index, end_index, ptLib, ptVol, generator, NbrMCsim

ptSwpt_current->swaptionMaturity = ptBermSwpt->swapMaturity - tenor; // Last e
time_index = end_index;

// At the last exercice date, price of the option = payoff.
for (m = 0; m < NbrMCsimulation; m++)
  {
    pnl_mat_get_row(ptLib_current->libor, LiborPathsMatrix, time_index + m * N
    LET(OptimalPayoff, m) = Swaption_Payoff_Discounted(ptLib_current, ptSwpt_c
  }

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

    // Explanatory variable
    for (m = 0; m < NbrMCsimulation; m++)
      {
        for (j = 0; j < Nfac; j++)
          {
            MLET(ExplicativeVariables, m, j) = MGET(BrownianMatrixPaths, time_
          }
      }

    // Least square fitting
    pnl_basis_fit_ls(basis, LS_RegressionCoeffVect, ExplicativeVariables, Opti

    pnl_mat_set_row(LS_RegressionCoeffMat, LS_RegressionCoeffVect, k - 1); //

    // Dynamical programing.
    for (m = 0; m < NbrMCsimulation; m++)
      {
        pnl_mat_get_row(ptLib_current->libor, LiborPathsMatrix, time_index + m
        payoff = Swaption_Payoff_Discounted(ptLib_current, ptSwpt_current, p,

        // If the payoff is null, the OptimalPayoff doesnt change.
```

```
          if (payoff > 0)
            {
              for (j = 0; j < Nfac; j++)
                {
                  VariablesExplicatives[j] = MGET(BrownianMatrixPaths, time_inde
                }

              regressed_value = pnl_basis_eval(basis, LS_RegressionCoeffVect, Va

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

  // The price at date 0 is the conditional expectation of OptimalPayoff, ie it'
  *LS_LowerPrice = pnl_vect_sum(OptimalPayoff) / NbrMCsimulation;

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

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

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


/** Upper bound for bermudan swaption using Andersen and Broadie algorithm.
 * @param SwaptionPriceUpper upper bound for the price on exit.
 * @param NbrMCsimulationDual number of outer simulation in Andersen and Broadie
 * @param NbrMCsimulationDualInternal  number of inner simulation in Andersen an
 * @param NbrMCsimulationPrimal number of simulation in Longstaff-Schwartz algor
 */
static void AndersenBroadie(double *SwaptionPriceUpper, double Nominal, long Nbr
```

```
{
  int j, m, m_i, N, k, Nfac, alpha, beta, Nsteps, save_all_paths, save_brownian;
  int NbrExerciseDates, start_index, end_index, nbr_var_explicatives, ExerciceOr
  double t, tenor, payoff, payoff_inner, numeraire_0, ContinuationValue, LowerPr
  double DoobMeyerMartingale, MaxVariable, Delta_0;
  double *VariablesExplicatives;


  PnlMat *LiborPathsMatrix, *BrownianMatrixPaths;
  PnlMat *LiborPathsMatrix_inner, *BrownianMatrixPaths_inner;
  PnlMat *LS_RegressionCoeffMat;
  PnlVect *LS_RegressionCoeffVect;
  PnlBasis *basis;

  Libor *ptLib_current;
  Libor *ptLib_inner;
  Swaption *ptSwpt_current_eur;
  Swaption *ptSwpt_current;

  Nfac = ptVol->numberOfFactors;
  N = ptLib->numberOfMaturities;
  tenor = ptBermSwpt->tenor;
  alpha = (int)(ptBermSwpt->swaptionMaturity / tenor); // T(alpha) is the swapti
  beta  = (int)(ptBermSwpt->swapMaturity / tenor); // T(beta) is the swap maturi
  NbrExerciseDates = beta - alpha;

  nbr_var_explicatives = Nfac;
  VariablesExplicatives = malloc(nbr_var_explicatives * sizeof(double));
  basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives);

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

  LiborPathsMatrix = pnl_mat_create(0, 0);
  BrownianMatrixPaths = pnl_mat_create(0, 0);
  LiborPathsMatrix_inner = pnl_mat_create(0, 0);
  BrownianMatrixPaths_inner = pnl_mat_create(0, 0);

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

```
numeraire_0 = Numeraire(0, ptLib, flag_numeraire);

// ptSwpt_current := le swap qui sera exerce à chaque date de la bermudeene. s
mallocSwaption(&ptSwpt_current, ptBermSwpt->swaptionMaturity, ptBermSwpt->swap
mallocSwaption(&ptSwpt_current_eur, ptBermSwpt->swaptionMaturity, ptBermSwpt->

// calcul de la borne inf du prix et des coefficients de regression.
MC_BermSwaption_LongstaffSchwartz(&LowerPrice_0, NbrMCsimulationPrimal, p, ptL

Delta_0 = 0;

save_brownian = 1;  // save_brownian = 1, we store the value brownian motion u
save_all_paths = 1; // save_all_paths = 0, we store only the simulated value o

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

Sim_Libor_Glasserman(start_index, end_index, ptLib, ptVol, generator, NbrMCsim

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

    pnl_mat_get_row(ptLib_current->libor, LiborPathsMatrix, start_index + m *
    ptSwpt_current->swaptionMaturity = ptBermSwpt->swaptionMaturity; // 1ere o
    payoff = Swaption_Payoff_Discounted(ptLib_current, ptSwpt_current, p, flag

    pnl_mat_get_row(LS_RegressionCoeffVect, LS_RegressionCoeffMat, 0);

    for (j = 0; j < Nfac; j++)
      {
        VariablesExplicatives[j] = MGET(BrownianMatrixPaths, start_index - 1 +
      }

    ContinuationValue = pnl_basis_eval(basis, LS_RegressionCoeffVect, Variable

    LowerPrice = MAX(ContinuationValue, payoff); // Prix d'apres Longstaff/Sch
    DoobMeyerMartingale = LowerPrice; // initialisation de la martingale utili
    LowerPriceOld = LowerPrice;
```

```
MaxVariable = payoff - DoobMeyerMartingale; // initialisation de la variab
for (k = 0; k < NbrExerciseDates - 2; k++)
  {
    start_index = alpha + k;
    end_index = start_index + 1;

    t = start_index * tenor;
    ptSwpt_current_eur->swaptionMaturity = t + tenor;

    /* numeraire_i = Numeraire(start_index, ptLib_current, flag_numeraire)
    //eur_swaption_price = (1./numeraire_i)*european_swaption_ap_rebonato(

    //ExerciceOrContinuation = (payoff>ContinuationValue && payoff>eur_swa
    ExerciceOrContinuation = payoff > ContinuationValue;

    pnl_mat_get_row(LS_RegressionCoeffVect, LS_RegressionCoeffMat, k + 1);
    ptSwpt_current->swaptionMaturity += tenor;

    // Si ExerciceOrContinuation=Exercice, on calcule l'esperance conditio
    if (ExerciceOrContinuation)
      {
        Sim_Libor_Glasserman(start_index, end_index, ptLib_current, ptVol,

        CondExpec_inner = 0;
        for (m_i = 0; m_i < NbrMCsimulationDualInternal; m_i++)
          {
            pnl_mat_get_row(ptLib_inner->libor, LiborPathsMatrix_inner, m_
            payoff_inner = Swaption_Payoff_Discounted(ptLib_inner, ptSwpt_

            for (j = 0; j < Nfac; j++)
              {
                VariablesExplicatives[j] = MGET(BrownianMatrixPaths,  star
              }

            ContinuationValue = pnl_basis_eval(basis, LS_RegressionCoeffVe

            CondExpec_inner += MAX(payoff_inner, ContinuationValue);
          }

        CondExpec_inner /= (double) NbrMCsimulationDualInternal;
```

```
      }

    // calcul du prix LS a la date T_(start_index+k+1)
    pnl_mat_get_row(ptLib_current->libor, LiborPathsMatrix, end_index + m
    payoff = Swaption_Payoff_Discounted(ptLib_current, ptSwpt_current, p,

    for (j = 0; j < Nfac; j++)
      {
        VariablesExplicatives[j] = MGET(BrownianMatrixPaths, start_index +
      }

    ContinuationValue = pnl_basis_eval(basis, LS_RegressionCoeffVect, Vari

    LowerPrice = MAX(payoff, ContinuationValue); // Prix(start_index+k+1)

    // Calcul de la martingale utilisee dans la borne sup du prix.
    if (ExerciceOrContinuation)
      {
        DoobMeyerMartingale = DoobMeyerMartingale + LowerPrice - CondExpec

      }
    else
      {
        DoobMeyerMartingale = DoobMeyerMartingale + LowerPrice - LowerPric
      }

    MaxVariable = MAX(MaxVariable, payoff - DoobMeyerMartingale);
    LowerPriceOld = LowerPrice;
  }

// Last Exercice Date. The price of the option here is equal to the payoff
start_index = beta - 2;
end_index = beta - 1;

t = start_index * tenor;
ptSwpt_current_eur->swaptionMaturity = t + tenor;

/* numeraire_i = Numeraire(start_index, ptLib_current, flag_numeraire); */
//eur_swaption_price = (1./numeraire_i)*european_swaption_ap_rebonato(t, p

//ExerciceOrContinuation = (payoff>ContinuationValue && payoff>eur_swaptio
```

```
    ExerciceOrContinuation = payoff > ContinuationValue;

    ptSwpt_current->swaptionMaturity += tenor; // derniere opportunite d'exerc

    if (ExerciceOrContinuation) //Si ExerciceOrContinuation==Exercice, on calc
      {
        Sim_Libor_Glasserman(start_index, end_index, ptLib_current, ptVol, gen

        CondExpec_inner = 0;
        for (m_i = 0; m_i < NbrMCsimulationDualInternal; m_i++)
          {
            pnl_mat_get_row(ptLib_inner->libor, LiborPathsMatrix_inner, m_i);
            payoff_inner = Swaption_Payoff_Discounted(ptLib_inner, ptSwpt_curr

            CondExpec_inner += payoff_inner; // le prix a la derniere opportun
          }

        CondExpec_inner /= NbrMCsimulationDualInternal;
      }

    pnl_mat_get_row(ptLib_current->libor, LiborPathsMatrix, end_index + m * Ns
    payoff = Swaption_Payoff_Discounted(ptLib_current, ptSwpt_current, p, flag

    LowerPrice = payoff;

    if (ExerciceOrContinuation)
      {
        DoobMeyerMartingale = DoobMeyerMartingale + LowerPrice - CondExpec_inn
      }
    else
      {
        DoobMeyerMartingale = DoobMeyerMartingale + LowerPrice - LowerPriceOld
      }

    MaxVariable = MAX(MaxVariable, payoff - DoobMeyerMartingale);

    Delta_0 += MaxVariable; // somme de MonteCarlo
  }

Delta_0 /= NbrMCsimulationDual;
```

```
  *SwaptionPriceUpper = numeraire_0 * Nominal * (LowerPrice_0 + 0.5 * Delta_0);

  free(VariablesExplicatives);
  pnl_basis_free(&basis);
  pnl_vect_free(&LS_RegressionCoeffVect);
  pnl_mat_free(&LS_RegressionCoeffMat);

  pnl_mat_free(&LiborPathsMatrix);
  pnl_mat_free(&BrownianMatrixPaths);
  pnl_mat_free(&LiborPathsMatrix_inner);
  pnl_mat_free(&BrownianMatrixPaths_inner);

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



static int MCAndersenBroadie(NumFunc_1 *p, double l0, double sigma_const, int nb
{
  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_maturity, 0.0, swaption_st

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

  AndersenBroadie(swaption_price_upper, Nominal, NbrMCsimulationDual, NbrMCsimul

  freeLibor(&ptLib);
  freeVolatility(&ptVol);
```

```
  freeSwaption(&ptBermSwpt);

  return init_mc;
}

int CALC(MC_AndersenBroadie_BermudanSwaption)(void *Opt, void *Mod, PricingMetho
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  return MCAndersenBroadie(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 - ptMod->T.Val.V_DATE,
                           ptOpt->OMaturity.Val.V_DATE - ptMod->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_LONG,
                           Met->Par[3].Val.V_ENUM.value,
                           Met->Par[4].Val.V_ENUM.value,
                           Met->Par[5].Val.V_INT,
                           Met->Par[6].Val.V_INT,
                           Met->Par[7].Val.V_ENUM.value,
                           &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(MC_AndersenBroadie_BermudanSwaption)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "PayerBermudanSwaption") == 0) || (strcmp((
    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 = 500;
      Met->Par[2].Val.V_LONG = 500;
      Met->Par[3].Val.V_ENUM.value = 0;
      Met->Par[3].Val.V_ENUM.members = &PremiaEnumMCRNGs;
      Met->Par[4].Val.V_ENUM.value = 0;
      Met->Par[4].Val.V_ENUM.members = &PremiaEnumBasis;
      Met->Par[5].Val.V_INT = 10;
      Met->Par[6].Val.V_INT = 1;
      Met->Par[7].Val.V_ENUM.value = 0;
      Met->Par[7].Val.V_ENUM.members = &PremiaEnumAfd;
    }

  return OK;
}


PricingMethod MET(MC_AndersenBroadie_BermudanSwaption) =
{
  "MC_AndersenBroadie_BermudanSwaption",
  {
    {"N iterations Primal", LONG, {100}, ALLOW},
    {"N iterations Dual", LONG, {100}, ALLOW},
    {"N iterations Dual internal", 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_AndersenBroadie_BermudanSwaption),
  {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(MC_AndersenBroadie_BermudanSwaption),
  CHK_ok,
  MET(Init)
};
```