Help

```
#include "
href../../../mod/lmm1d/lmm1d_exoi/lmm1d_exoi_h_src.pdflmm1d_exoi.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 < (2011+2) //The "#els
static int CHK_OPT(MC_LongstaffSchwartz_CallableCappedFloater)(void *Opt, void *
{
  return NONACTIVE;
}
int CALC(MC_LongstaffSchwartz_CallableCappedFloater)(void *Opt, void *Mod, Prici
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else
//double CallableContract_Payement(int i, Libor *ptLib, *PnlVect ContractParams,

double ProductPayment(char *CouponFlag, int i, Libor *ptLib, PnlVect *ContractPa
{
  int m1, m2;
  double tenor, P_i = 0., libor_i, numeraire_i, coupon_i = 0., CMSRate1 = 0., CM
  double Spread = 0., Cap = 0., Strike = 0., Gearing = 0., Floor = 0.;
  double FixedRate = 0., LowerRangeBound = 0., UpperRangeBound = 0.;
  double CMSMat1 = 0., CMSMat2 = 0.;

  tenor = ptLib->tenor;
  libor_i = GET(ptLib->libor, i);
  P_i = 1. / (1 + tenor * libor_i);
  numeraire_i = Numeraire(i, ptLib, flag_numeraire);

  if (strcmp(CouponFlag, "CallableCappedFloater") == 0)
    {
      Spread  = GET(ContractParams, 0);
      Cap     = GET(ContractParams, 1);
      coupon_i  = MIN(libor_i + Spread, Cap);
    }
```

```
  else if (strcmp(CouponFlag, "CallableInverseFloater") == 0)
    {
      Cap     = GET(ContractParams, 0);
      Strike  = GET(ContractParams, 1);
      Gearing = GET(ContractParams, 2);
      Floor   = GET(ContractParams, 3);
      coupon_i  = MIN(MAX(Strike - Gearing * libor_i, Floor), Cap);
    }

  else if (strcmp(CouponFlag, "CallableRangeAccrual") == 0)
    {
      FixedRate       = GET(ContractParams, 0);
      LowerRangeBound = GET(ContractParams, 1);
      UpperRangeBound = GET(ContractParams, 2);
      coupon_i          = FixedRate * (libor_i <= UpperRangeBound) * (libor_i >=
    }

  else if (strcmp(CouponFlag, "CallableCMSSpread") == 0)
    {
      Cap         = GET(ContractParams, 0);
      Floor       = GET(ContractParams, 1);
      CMSMat1     = GET(ContractParams, 2);
      CMSMat2     = GET(ContractParams, 3);

      m1 = pnl_iround(CMSMat1 / tenor);
      m2 = pnl_iround(CMSMat2 / tenor);

      CMSRate1    = computeSwapRate(ptLib, i, i, m1);
      CMSRate2    = computeSwapRate(ptLib, i, i, m2);

      coupon_i      = MAX(MIN(CMSRate1 - CMSRate2, Cap), Floor);
    }

  // The - sign is used because we estimate cancelable and non-callable contract
  return P_i * tenor * (coupon_i - libor_i) / numeraire_i;
}

void MC_ExoticProduct_LongstaffSchwartz(char *CouponFlag, PnlVect *ContractParam
{
  int alpha, beta, m, k, N, NbrExerciseDates, time_index, save_brownian, save_al
```

```
double tenor, regressed_value, payoff_approx, numeraire_0;
double *VariablesExplicatives;

Libor *ptL_current;
PnlMat *LiborPathsMatrix, *BrownianMatrixPaths;
PnlMat *ExplicativeVariables;
PnlVect *OptimalPayoff, *CurrentPayoff;
PnlVect *RegCoeffVect_optimal, *RegCoeffVect_current;
PnlBasis *basis;

//Nfac = ptVol->numberOfFactors;
N = ptLib->numberOfMaturities;
tenor = ptLib->tenor;
alpha = pnl_iround(first_exercise_date / tenor);
beta  = pnl_iround(last_payement_date / tenor);
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 * sizeof(double));
ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nbr_var_explicatives);
OptimalPayoff = pnl_vect_create(NbrMCsimulation);
CurrentPayoff = pnl_vect_create(NbrMCsimulation);
RegCoeffVect_optimal = pnl_vect_new();
RegCoeffVect_current = pnl_vect_new();
LiborPathsMatrix = pnl_mat_new(); // LiborPathsMatrix contains all the traject
BrownianMatrixPaths = pnl_mat_new(); // We store also the brownian values to b

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives);

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

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
```

```
// At the last exercice date, price of the option = payoff.
time_index = end_index;
for (m = 0; m < NbrMCsimulation; m++)
  {
    pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix, time_index + m * Nst

    LET(OptimalPayoff, m) = MAX(0., ProductPayment(CouponFlag, time_index, ptL
  }
pnl_vect_clone(CurrentPayoff, OptimalPayoff);

for (k = NbrExerciseDates - 1; k >= 1; k--)
  {
    time_index -= 1;

    // Explanatory variable
    for (m = 0; m < NbrMCsimulation; m++)
      {
        pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix, time_index + m *
        MLET(ExplicativeVariables, m, 0) = computeSwapRate(ptL_current, time_i
        MLET(ExplicativeVariables, m, 1) = GET(ptL_current->libor, time_index)

        LET(CurrentPayoff, m) += ProductPayment(CouponFlag, time_index, ptL_cu
      }

    // Least square fitting
    pnl_basis_fit_ls(basis, RegCoeffVect_current, ExplicativeVariables, Curren
    pnl_basis_fit_ls(basis, RegCoeffVect_optimal, ExplicativeVariables, Optima

    // Equation de programmation dynamique.
    for (m = 0; m < NbrMCsimulation; m++)
      {
        pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix, time_index + m *
        VariablesExplicatives[0] = computeSwapRate(ptL_current, time_index, ti
        VariablesExplicatives[1] = GET(ptL_current->libor, time_index);

        payoff_approx = pnl_basis_eval(basis, RegCoeffVect_current, VariablesE

        // If the payoff is null, the OptimalPayoff doesn't change.
        if (payoff_approx > 0)
          {
```

```
              regressed_value = pnl_basis_eval(basis, RegCoeffVect_optimal, Vari
              if (payoff_approx > regressed_value)
                {
                  LET(OptimalPayoff, m) = payoff_approx;
                }
            }
        }
    }

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

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

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

  pnl_vect_free(&CurrentPayoff);
  pnl_vect_free(&OptimalPayoff);
  pnl_vect_free(&RegCoeffVect_current);
  pnl_vect_free(&RegCoeffVect_optimal);
  pnl_mat_free(&BrownianMatrixPaths);

  freeLibor(&ptL_current);
}


static int MCLongstaffSchwartz(double l0, double sigma_const, int nb_factors, do
{
  Volatility *ptVol;
  Libor *ptLib;
  int init_mc;
  int Nbr_Maturities;
  char *CouponFlag = "CallableCappedFloater";
  PnlVect *ContractParams = pnl_vect_create(2);

  LET(ContractParams, 0) = spread_rate;
  LET(ContractParams, 1) = cap_rate;
```

```
  Nbr_Maturities = pnl_iround(last_payement_date / tenor);

  mallocLibor(&ptLib , Nbr_Maturities, tenor, l0);
  mallocVolatility(&ptVol , nb_factors, sigma_const);

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

  MC_ExoticProduct_LongstaffSchwartz(CouponFlag, ContractParams, swaption_price,

  freeLibor(&ptLib);
  freeVolatility(&ptVol);
  pnl_vect_free(&ContractParams);

  return init_mc;
}

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

  return MCLongstaffSchwartz(ptMod->l0.Val.V_PDOUBLE,
                             ptMod->Sigma.Val.V_PDOUBLE,
                             ptMod->NbFactors.Val.V_ENUM.value,
                             ptOpt->LastPaymentDate.Val.V_DATE - ptMod->T.Val.V_
                             ptOpt->FirstExerciseDate.Val.V_DATE - ptMod->T.Val.
                             ptOpt->Nominal.Val.V_PDOUBLE,
                             ptOpt->Cap.Val.V_PDOUBLE,
                             ptOpt->Spread.Val.V_PDOUBLE,
                             ptOpt->ResetPeriod.Val.V_DATE,
                             Met->Par[0].Val.V_LONG,
                             Met->Par[1].Val.V_ENUM.value,
                             Met->Par[2].Val.V_ENUM.value,
                             Met->Par[3].Val.V_INT,
                             Met->Par[4].Val.V_INT,
                             Met->Par[5].Val.V_ENUM.value,
                             &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(MC_LongstaffSchwartz_CallableCappedFloater)(void *Opt, void *
```

```
{
  if ((strcmp(((Option *)Opt)->Name, "CallableCappedFloater") == 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_CallableCappedFloater) =
{
  "MC_LongstaffSchwartz_Callable_Capped_Floater",
  {
    {"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_CallableCappedFloater),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_LongstaffSchwartz_CallableCappedFloater),
  CHK_ok,
  MET(Init)
};
```