```
   Help
#include "lmm1d_exoi.h"
#include "pnl/pnl_basis.h"
#include "math/mc_lmm_glassermanzhao.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2011+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_CallableCappedFloa
    ter)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_LongstaffSchwartz_CallableCappedFloater)(void *
    Opt,void *Mod,PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else
//double CallableContract_Payement(int i, Libor *ptLib, *Pn
    lVect ContractParams, int flag_numeraire);

double ProductPayment(char* CouponFlag, int i, Libor *pt
    Lib, PnlVect *ContractParams, int flag_numeraire)
{
  int m1, m2;
  double tenor, P_i=0., libor_i, numeraire_i, coupon_i=0.,
    CMSRate1=0., CMSRate2=0.;
  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<=UpperRangeBou
  nd)*(libor_i>=LowerRangeBound);
  }

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

    m1 = intapprox(CMSMat1/tenor);
    m2 = intapprox(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 contracts values then deduce callable contract
    value.
  return P_i*tenor*(coupon_i - libor_i)/numeraire_i;
}

void MC_ExoticProduct_LongstaffSchwartz(char* CouponFlag,
    PnlVect *ContractParams, double *LS_Price, double first_exe
    rcise_date, double last_payement_date, double Nominal, int
    NbrMCsimulation, Libor *ptLib, Volatility *ptVol, int     generator, int bas
    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_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 = intapprox(first_exercise_date/tenor);
  beta  = intapprox(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*size
    of(double));
```

```
ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nb
  r_var_explicatives); // Explicatives variables
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 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);

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);

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

    LET(OptimalPayoff, m) = MAX(0., ProductPayment(
  CouponFlag, time_index, ptL_current, ContractParams, flag_numera
  ire));
  }
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, 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);

      LET(CurrentPayoff, m) += ProductPayment(CouponF
lag, time_index, ptL_current, ContractParams, flag_numerair
e);
    }

  // Least square fitting
  pnl_basis_fit_ls(basis, RegCoeffVect_current, Explic
ativeVariables, CurrentPayoff);
  pnl_basis_fit_ls(basis, RegCoeffVect_optimal, Explic
ativeVariables, OptimalPayoff);

  // Equation de programmation dynamique.
  for (m=0; m<NbrMCsimulation; m++)
    {
      pnl_mat_get_row(ptL_current->libor, LiborPathsM
atrix, time_index + m*Nsteps);
      VariablesExplicatives[0] = computeSwapRate(ptL_
current, time_index, time_index, beta);
      VariablesExplicatives[1] = GET(ptL_current->
libor, time_index);

      payoff_approx = pnl_basis_eval(basis, RegCoeffVec
t_current, VariablesExplicatives);

      // If the payoff is null, the OptimalPayoff doesn
't change.
      if (payoff_approx>0)
        {
          regressed_value = pnl_basis_eval(basis, Reg
CoeffVect_optimal, VariablesExplicatives);
          if (payoff_approx > regressed_value)
```

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

  // 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);

  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_cons
    t, int nb_factors, double last_payement_date, double first_
    exercise_date,  double Nominal, double cap_rate, double spr
    ead_rate, double tenor, long NbrMCsimulation, int     generator, int basis_n
    flag_numeraire, double *swaption_price)
{
  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 = intapprox(last_payement_date/tenor);

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

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

  MC_ExoticProduct_LongstaffSchwartz(CouponFlag, ContractP
    arams, swaption_price, first_exercise_date, last_payement_
    date, Nominal, NbrMCsimulation, ptLib, ptVol, generator,
    basis_name, DimApprox, NbrStepPerTenor, flag_numeraire);

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

  return init_mc;
}

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

  return MCLongstaffSchwartz(    ptMod->l0.Val.V_PDOUBLE,
                                 ptMod->Sigma.Val.V_PDOUB
    LE,
                                 ptMod->NbFactors.Val.V_
    ENUM.value,
                                 ptOpt->LastPaymentDate.Val
    .V_DATE-ptMod->T.Val.V_DATE,
                                 ptOpt->FirstExerciseDate.
    Val.V_DATE-ptMod->T.Val.V_DATE,
                                 ptOpt->Nominal.Val.V_PDOUB
    LE,
```

```
                                    ptOpt->Cap.Val.V_PDOUBLE,
                                    ptOpt->Spread.Val.V_PDOUB
    LE,
                                    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_CallableCappedFloa
    ter)(void *Opt, void *Mod)
{
  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_CallableCappedFloa
    ter)=
{
  "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)
};
```

References