Help

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

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(MC_Andersen_BermudanSwaption)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_Andersen_BermudanSwaption)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


/// The exercise strategy, proposed by Andersen(1999), for a bermudan swaption w
/// At exercice date T(i), we exercise if the swaption intrinsic value (stored i
/// These parameters are stored in vector AndersenParams. For more flexibility,
/// To choose these parameters, we maximize the price of bermudan swaptions. We

/** Structure that contains information about Andersen exercise strategy. **/
typedef struct
{
  int NbrExerciseDates;
  int NbrMCsimulation;
  int j_start; // Index of current exercise date where parameter will be estimat
  int q; // Number of kink-points

  double H_max; // Maximum value of parameters, used in the minimization routine

  PnlMat *DiscountedPayoff; // Matrix containing the swaption discounted payoff
  PnlMat *NumeraireValue; // Matrix containing value of the numeraire considered
  PnlVect *AndersenParams; // Vector containing parameters that define the Ander
```

```
  PnlVectInt *AndersenIndices; // Indices of kink-points where we compute the pa

} AndersenStruct;

static int Create_AndersenStruct(AndersenStruct *andersen_struct)
{
  andersen_struct->DiscountedPayoff = pnl_mat_create(0, 0);
  andersen_struct->NumeraireValue = pnl_mat_create(0, 0);
  andersen_struct->AndersenParams = pnl_vect_create(0);
  andersen_struct->AndersenIndices = pnl_vect_int_create(0);

  return OK;
}

static int Free_AndersenStruct(AndersenStruct *andersen_struct)
{
  pnl_mat_free(&(andersen_struct->DiscountedPayoff));
  pnl_mat_free(&(andersen_struct->NumeraireValue));
  pnl_vect_free(&(andersen_struct->AndersenParams));
  pnl_vect_int_free(&(andersen_struct->AndersenIndices));

  return OK;
}

// Initialization of AndersenStruct. We fill the matrices DiscountedPayoff and N
// We also fill AndersenIndices with "q" kink-points.
static int Init_AndersenStruct(AndersenStruct *andersen_struct, Libor *ptLib, Sw
{
  int alpha, beta, start_index, end_index, save_brownian, save_all_paths;
  int i, m, j, NbrExerciseDates, step;
  double tenor, param_max, discounted_payoff_j, numeraire_j;

  Libor *ptL_current;
  Swaption *ptSwpt;

  PnlMat *LiborPathsMatrix;
  LiborPathsMatrix = pnl_mat_create(0, 0);

  tenor = ptBermSwpt->tenor;
  alpha = pnl_iround(ptBermSwpt->swaptionMaturity / tenor); // T(alpha) is the s
  beta  = pnl_iround(ptBermSwpt->swapMaturity / tenor); // T(beta) is the swap m
```

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

param_max = 0;
save_brownian = 0;
save_all_paths = 1;

// SImulation of "NbrMCsimulation" Libor paths under "flag_numeraire" measure.
Sim_Libor_Glasserman(start_index, end_index, ptLib, ptVol, generator, NbrMCsim

step = (NbrExerciseDates - 1) / q;

mallocLibor(&ptL_current, LiborPathsMatrix->n, tenor, 0.);
mallocSwaption(&ptSwpt, ptBermSwpt->swaptionMaturity, ptBermSwpt->swapMaturity

andersen_struct->NbrExerciseDates = NbrExerciseDates;
andersen_struct->NbrMCsimulation = NbrMCsimulation;
andersen_struct->j_start = 0;
andersen_struct->q = q;

pnl_mat_resize(andersen_struct->DiscountedPayoff, NbrExerciseDates, NbrMCsimul
pnl_mat_resize(andersen_struct->NumeraireValue, NbrExerciseDates, NbrMCsimulat
pnl_vect_resize(andersen_struct->AndersenParams, NbrExerciseDates);
pnl_vect_int_resize(andersen_struct->AndersenIndices, q + 1);

// Set the indices of kink-points, where parmeters will be estimated
pnl_vect_int_set(andersen_struct->AndersenIndices, q, NbrExerciseDates - 1);
pnl_vect_int_set(andersen_struct->AndersenIndices, 0, 0);
for (i = 1; i < q; i++)
  {
    pnl_vect_int_set(andersen_struct->AndersenIndices, q - i, (NbrExerciseDate
  }

// Fill the structure andersen_struct with discounted payoff and numeraire val
for (j = alpha; j < beta; j++)
  {
    for (m = 0; m < NbrMCsimulation; m++)
      {
        pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix, j + m * end_inde
```

```
          discounted_payoff_j = Nominal * Swaption_Payoff_Discounted(ptL_current
          numeraire_j = Numeraire(j, ptL_current, flag_numeraire);

          MLET(andersen_struct->DiscountedPayoff, j - alpha, m) = discounted_pay
          MLET(andersen_struct->NumeraireValue, j - alpha, m) = numeraire_j;

          param_max = MAX(param_max, numeraire_j * discounted_payoff_j);
        }

      ptSwpt->swaptionMaturity += tenor;
    }

  andersen_struct->H_max = param_max;

  pnl_mat_free(&LiborPathsMatrix);
  freeSwaption(&ptSwpt);
  freeLibor(&ptL_current);

  return OK;
}

// We interpolate linearly the parameters of exercise strategy between intermidi
static int Interpolate_AndersenParams(AndersenStruct *andersen_struct)
{
  int i, j, q, j1, j2;
  double a, b;

  q = andersen_struct->q;

  for (i = 0; i < q; i++)
    {
      j1 = pnl_vect_int_get(andersen_struct->AndersenIndices, i);
      j2 = pnl_vect_int_get(andersen_struct->AndersenIndices, i + 1);

      for (j = j1; j <= j2; j++)
        {
          a = ((double)(j - j1)) / ((double)(j2 - j1));
          b = ((double)(j2 - j)) / ((double)(j2 - j1));

          LET(andersen_struct->AndersenParams, j) = a * GET(andersen_struct->And
        }
```

```
    }

  return OK;
}

// This function computes the prices of bermudan swaption corresponding to the e
static double AmOption_Price_Andersen(AndersenStruct *andersen_struct)
{
  long NbrMCsimulation;
  int NbrExerciseDates, j, j_start, m;
  double andersen_param, discounted_payoff, mean_estim, numeraire_j, PriceBermSw

  Interpolate_AndersenParams(andersen_struct);

  j_start = andersen_struct->j_start;
  NbrExerciseDates = andersen_struct->NbrExerciseDates;
  NbrMCsimulation = andersen_struct->NbrMCsimulation;

  mean_estim = 0.;
  for (m = 0; m < NbrMCsimulation; m++)
    {
      j = j_start;
      do
        {
          discounted_payoff = MGET(andersen_struct->DiscountedPayoff, j, m);
          numeraire_j = MGET(andersen_struct->NumeraireValue, j, m);
          andersen_param = GET(andersen_struct->AndersenParams, j);
          j++;
        }
      while (discounted_payoff * numeraire_j <= andersen_param && j < NbrExercis

      mean_estim += discounted_payoff;
    }

  PriceBermSwp = mean_estim / (double)NbrMCsimulation;

  return PriceBermSwp;
}

// Scalar function to be minimized in order to get optimal parameters.
```

```
static double func_to_minimize(double x, void *andersen_struct)
{
  LET(((AndersenStruct *)andersen_struct)->AndersenParams, ((AndersenStruct *)an

  return -AmOption_Price_Andersen(andersen_struct);

}

// Compute the price of a bermudan swaption.
static int MC_BermSwpaption_Andersen(NumFunc_1 *p, Libor *ptLib, Swaption *ptBer
{
  int alpha, beta, i, NbrExerciseDates;
  double tenor, numeraire_0;
  double ax, bx,  cx, tol, xmin;

  AndersenStruct andersen_struct;
  PnlFunc FuncToMinimize;

  Create_AndersenStruct(&andersen_struct);

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


  numeraire_0 = Numeraire(0, ptLib, flag_numeraire);

  tol = 1e-10;
  q = MIN(q, NbrExerciseDates - 1); // The maximum number of kink-points that ca
  q = MAX(1, q); // q must be greater than zero.

  FuncToMinimize.F = &func_to_minimize;
  FuncToMinimize.params = &andersen_struct;

  // Initialize the structure andersen_struct using "NbrMCsimulation_param" path
  // We will use these paths to estimates the optimal parameters of exercise str
  Init_AndersenStruct(&andersen_struct, ptLib, ptBermSwpt, ptVol, p, NbrMCsimula
```

```
  // At maturity, the parameter is null, because we exercise whenever payoff is
  pnl_vect_set_zero(andersen_struct.AndersenParams);

  ax = 0; // lower point for GoldenSearch method
  cx = andersen_struct.H_max; // upper point for GoldenSectionSearch method
  bx = 0.5 * (ax + cx); // middle point for GoldenSectionSearch method

  for (i = q - 1; i >= 0; i--)
    {
      // Index of exercise date where we compute parameter of exercise strategy.
      andersen_struct.j_start = pnl_vect_int_get(andersen_struct.AndersenIndices

      // Find optimal parameter at current exercise date.
      golden(&FuncToMinimize, ax, bx, cx, tol, &xmin);

      // Store this parameter in AndersenParams.
      LET(andersen_struct.AndersenParams, andersen_struct.j_start) = xmin;

      ax = 0.5 * xmin;
      bx = 0.5 * (ax + cx);
    }

  // We simulate another set of Libor paths, independants of the ones used to es
  // In general, choose NbrMCsimulation >> NbrMCsimulation_param.
  Init_AndersenStruct(&andersen_struct, ptLib, ptBermSwpt, ptVol, p, NbrMCsimula

  // Finaly, we use the found parameters to estimate the price of bermudan swapt
  andersen_struct.j_start = 0;
  *PriceBermSwp = numeraire_0 * AmOption_Price_Andersen(&andersen_struct);

  // Free memory.
  Free_AndersenStruct(&andersen_struct);

  return OK;
}

static int MC_BermSwpaption_LMM_Andersen(NumFunc_1 *p, double l0, double sigma_c
{
  Volatility *ptVol;
  Libor *ptLib;
  Swaption *ptBermSwpt;
```

```
  int init_mc;
  int Nbr_Maturities;

  Nbr_Maturities = pnl_iround(swap_maturity / tenor);

  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, NbrMCsimulation);
  if (init_mc != OK) return init_mc;

  MC_BermSwpaption_Andersen(p, ptLib, ptBermSwpt, ptVol, Nominal, NbrMCsimulatio

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

  return init_mc;
}


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

  return MC_BermSwpaption_LMM_Andersen(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.Va
                                       ptOpt->OMaturity.Val.V_DATE - ptMod->T.Va
                                       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_INT,
                                       Met->Par[3].Val.V_ENUM.value,
                                       Met->Par[4].Val.V_ENUM.value,
```

```
                                        Met->Par[5].Val.V_INT,
                                        &(Met->Res[0].Val.V_DOUBLE));

}

static int CHK_OPT(MC_Andersen_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 = 10000;
      Met->Par[1].Val.V_LONG = 50000;
      Met->Par[2].Val.V_INT = 1;
      Met->Par[3].Val.V_ENUM.value = 0;
      Met->Par[3].Val.V_ENUM.members = &PremiaEnumRNGs;
      Met->Par[4].Val.V_ENUM.value = 0;
      Met->Par[4].Val.V_ENUM.members = &PremiaEnumAfd;
      Met->Par[5].Val.V_INT = 3;
    }

  return OK;
}


PricingMethod MET(MC_Andersen_BermudanSwaption) =
{
  "MC_Andersen_BermudanSwaption",
  {
    {"N Simulations Parms", LONG, {100}, ALLOW},
    {"N Simulations", LONG, {100}, ALLOW},
    {"N Steps per Period", INT, {100}, ALLOW},
```

```
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Martingale Measure", ENUM, {100}, ALLOW},
    {"N Kink-Points", INT, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_Andersen_BermudanSwaption),
  {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(MC_Andersen_BermudanSwaption),
  CHK_ok,
  MET(Init)
};
```