Help

```
#include "lmm1d_stdi.h"
#include "math/mc_lmm_glassermanzhao.h"
#include "math/golden.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2007+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_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 with N exercise dates is determined by
    a vector of N parameters.
/// At exercice date T(i), we exercise if the swaption intr
    insic value (stored in DiscountedPayoff, in discounted shape
    ) is greater than a deterministic parameter.
/// These parameters are stored in vector AndersenParams.
    For more flexibility, we estimate parameters only on a su
    bset of exercise dates, then interpolate them linearly, as
    proposed in Andersen article(1999).
/// To choose these parameters, we maximize the price of
    bermudan swaptions. We then get a sub-optimal strategy that
    gives a lower bound for the true price.

/** Structure that contains information about Andersen exe
    rcise strategy. **/
typedef struct
{
  int NbrExerciseDates;
```

```
  int NbrMCsimulation;
  int j_start; // Index of current exercise date where para
    meter will be estimated.
  int q; // Number of kink-points

  double H_max; // Maximum value of parameters, used in th
    e minimization routine.

  PnlMat *DiscountedPayoff; // Matrix containing the swapt
    ion discounted payoff at each exercise date
  PnlMat *NumeraireValue; // Matrix containing value of th
    e numeraire considered in the simulation of Libor rates.
  PnlVect *AndersenParams; // Vector containing parameters
    that define the Andersen strategy
  PnlVectInt *AndersenIndices; // Indices of kink-points wh
    ere we compute the parameters of exercise strategy. For the
    rest of them, we interpolate.

} 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 NumeraireValue using simulated paths.
// We also fill AndersenIndices with "q" kink-points.
static int Init_AndersenStruct(AndersenStruct *andersen_
    struct, Libor *ptLib, Swaption *ptBermSwpt, Volatility *ptVol,
    NumFunc_1 *p, int NbrMCsimulation, int NbrStepPerTenor, int    generator, i
{
  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 = intapprox(ptBermSwpt->swaptionMaturity/tenor); //
     T(alpha) is the swaption maturity
  beta  = intapprox(ptBermSwpt->swapMaturity/tenor); // T(
    beta) is the swap maturity
  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 "fla
    g_numeraire" measure.
  Sim_Libor_Glasserman(start_index, end_index, ptLib, pt    Vol, generator, NbrM
    paths, LiborPathsMatrix, save_brownian, LiborPathsMatrix, fla
    g_numeraire);

  step = (NbrExerciseDates-1)/q;
```

```
mallocLibor(&ptL_current, LiborPathsMatrix->n, tenor, 0.)
  ;
mallocSwaption(&ptSwpt, ptBermSwpt->swaptionMaturity, pt
  BermSwpt->swapMaturity, 0.0, ptBermSwpt->strike, tenor);

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

pnl_mat_resize(andersen_struct->DiscountedPayoff, NbrEx
  erciseDates, NbrMCsimulation);
pnl_mat_resize(andersen_struct->NumeraireValue, NbrExerc
  iseDates, NbrMCsimulation);
pnl_vect_resize(andersen_struct->AndersenParams, NbrExerc
  iseDates);
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, Nb
  rExerciseDates-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, (NbrExerciseDates-1)-i*step);
  }

// Fill the structure andersen_struct with discounted
  payoff and numeraire values.
for (j=alpha; j<beta; j++)
  {
    for (m=0; m<NbrMCsimulation; m++)
      {
        pnl_mat_get_row(ptL_current->libor, LiborPathsM
  atrix, j + m*end_index);

        discounted_payoff_j = Nominal*Swaption_Payoff_Dis
```

```
    counted(ptL_current, ptSwpt, p, flag_numeraire);
          numeraire_j = Numeraire(j, ptL_current, flag_
    numeraire);

          MLET(andersen_struct->DiscountedPayoff, j-alpha,
    m) = discounted_payoff_j;
          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 stra
    tegy between intermidiate exercise dates.
static int Interpolate_AndersenParams(AndersenStruct *ande
    rsen_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->AndersenIndic
    es, i);
      j2 = pnl_vect_int_get(andersen_struct->AndersenIndic
    es, 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->AndersenParams, j2) + b*GET(andersen_
    struct->AndersenParams, j1);
        }

    }

  return OK;
}

// This function computes the prices of bermudan swaption
    corresponding to the exercise strategy defined by andersen_
    struct.
static double AmOption_Price_Andersen(AndersenStruct *ande
    rsen_struct)
{
  long NbrMCsimulation;
  int NbrExerciseDates, j, j_start, m;
  double andersen_param, discounted_payoff, mean_estim,
    numeraire_j, PriceBermSwp;

  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->Discoun
    tedPayoff, j, m);
```

```
        numeraire_j = MGET(andersen_struct->NumeraireValu
    e, j, m);
        andersen_param = GET(andersen_struct->AndersenPar
    ams, j);
        j++;
      }
      while (discounted_payoff*numeraire_j <=andersen_para
    m && j<NbrExerciseDates);

      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*)andersen_struct)->j_start) = x;

  return -AmOption_Price_Andersen(andersen_struct);

}

// Compute the price of a bermudan swaption.
static int MC_BermSwpaption_Andersen(NumFunc_1 *p, Libor *
    ptLib, Swaption *ptBermSwpt, Volatility *ptVol, double Nom
    inal, long NbrMCsimulation_param, long NbrMCsimulation, int
    NbrStepPerTenor, int generator, int flag_numeraire, int q,
    double *PriceBermSwp)
{
  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 = intapprox(ptBermSwpt->swaptionMaturity/tenor); //
   T(alpha) is the swaption maturity
beta  = intapprox(ptBermSwpt->swapMaturity/tenor); // T(
  beta) is the swap maturity
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 can be used is NbrExerciseDates-1.
q = MAX(1, q); // q must be greater than zero.

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

// Initialize the structure andersen_struct using "NbrMCs
  imulation_param" paths.
// We will use these paths to estimates the optimal para
  meters of exercise strategy.
Init_AndersenStruct(&andersen_struct, ptLib, ptBermSwpt,
  ptVol, p, NbrMCsimulation_param, NbrStepPerTenor,     generator, flag_numera

// At maturity, the parameter is null, because we exercis
  e whenever payoff is positif.
pnl_vect_set_zero(andersen_struct.AndersenParams);

ax = 0; // lower point for GoldenSearch method
cx = andersen_struct.H_max; // upper point for GoldenSec
  tionSearch method
bx = 0.5*(ax+cx); // middle point for GoldenSectionSearc
  h 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, i);

      // 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 estimate the parameters of exercise
    strategy.
  // In general, choose NbrMCsimulation >> NbrMCsimulation_
    param.
  Init_AndersenStruct(&andersen_struct, ptLib, ptBermSwpt,
    ptVol, p, NbrMCsimulation, NbrStepPerTenor, generator, fla
    g_numeraire, Nominal, q);

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

  // Free memory.
  Free_AndersenStruct(&andersen_struct);

  return OK;
}

static int MC_BermSwpaption_LMM_Andersen(NumFunc_1 *p,
    double l0, double sigma_const, int nb_factors, double swap_matu
```

```
    rity, double swaption_maturity,  double Nominal, double swa
    ption_strike, double tenor, long NbrMCsimulation_param, lon
    g NbrMCsimulation, int NbrStepPerTenor, int generator, int
    flag_numeraire, int q, double *swaption_price)
{
  Volatility *ptVol;
  Libor *ptLib;
  Swaption *ptBermSwpt;
  int init_mc;
  int Nbr_Maturities;

  Nbr_Maturities = intapprox(swap_maturity/tenor);

  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, NbrMCsimulat
    ion);
  if (init_mc != OK) return init_mc;

  MC_BermSwpaption_Andersen(p, ptLib, ptBermSwpt, ptVol,
    Nominal, NbrMCsimulation_param, NbrMCsimulation, NbrStepPe
    rTenor, generator, flag_numeraire, q, swaption_price);

  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.Val.V_DATE,
                                        ptOpt->OMaturity.
    Val.V_DATE-ptMod->T.Val.V_DATE,
                                        ptOpt->Nominal.
    Val.V_PDOUBLE,
                                        ptOpt->FixedRate.
    Val.V_PDOUBLE,
                                        ptOpt->ResetPerio
    d.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(((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=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)
};
```

References