Help

```
#include  "lmm1d_stdi.h"
#include "pnl/pnl_basis.h"
#include "math/mc_lmm_glassermanzhao.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+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_AndersenBroadie_BermudanSwaption)(voi
    d *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_AndersenBroadie_BermudanSwaption)(void *Opt,voi
    d *Mod,PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/** Lower bound for bermudan swaption using Longstaff-Schwa
    rtz algorithm
 * We store the regression coefficients in a matrix LS_Reg
    ressionCoeffMat
 * @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 function
 * @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+1)
 * @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 coefficients
 * Rmk: Libor rates are simulated using the method proposed
    by Glasserman-Zhao.
 */
static void MC_BermSwaption_LongstaffSchwartz(double *LS_
   LowerPrice, int NbrMCsimulation, NumFunc_1 *p, Libor *ptLib,
    Swaption *ptBermSwpt, Volatility *ptVol, int generator,
   int basis_name, int DimApprox, int NbrStepPerTenor, int flag_
   numeraire, PnlMat *LS_RegressionCoeffMat)
{
  int alpha, beta, j, m, k, N, NbrExerciseDates, time_ind
    ex, save_brownian, save_all_paths, start_index, end_index,
    Nsteps, nbr_var_explicatives, Nfac;
  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 swaption maturity
  beta  = (int)(ptBermSwpt->swapMaturity/tenor); // T(beta)
     is the swap maturity
  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*size
  of(double));
ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nb
  r_var_explicatives); // Explicatives variables
OptimalPayoff = pnl_vect_create(NbrMCsimulation);
LS_RegressionCoeffVect = pnl_vect_create(0);
LiborPathsMatrix = pnl_mat_create(0, 0); // LiborPathsM
  atrix contains all the trajectories.
BrownianMatrixPaths = pnl_mat_create(0, 0); // We store
  also the brownian values to be used a explicatives variables.

pnl_mat_resize(LS_RegressionCoeffMat, NbrExerciseDates-1,
   DimApprox);

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_e
  xplicatives);

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

// ptSwpt_current := contains the information about the
  swap to be be exerced at each exercice date.
// The maturity of the swap stays the same.
mallocSwaption(&ptSwpt_current, ptBermSwpt->swaptionMatu
  rity, ptBermSwpt->swapMaturity, 0.0, ptBermSwpt->strike, ten
  or);

//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);

ptSwpt_current->swaptionMaturity = ptBermSwpt->swapMatu
  rity - tenor; // Last exerice date.
time_index = end_index;
```

```
// At the last exercice date, price of the option = payo
  ff.
for (m=0; m<NbrMCsimulation; m++)
  {
    pnl_mat_get_row(ptLib_current->libor, LiborPathsMatr
  ix, time_index + m*Nsteps);
    LET(OptimalPayoff, m) = Swaption_Payoff_Discounted(pt
  Lib_current, ptSwpt_current, p, flag_numeraire);
  }

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(Brow
  nianMatrixPaths, time_index-1 + m*Nsteps, j);
          }
      }

    // Least square fitting
    pnl_basis_fit_ls(basis,LS_RegressionCoeffVect, Explic
  ativeVariables, OptimalPayoff);

    pnl_mat_set_row(LS_RegressionCoeffMat, LS_Regression
  CoeffVect, k-1); // Store regression coefficients

    // Dynamical programing.
    for (m=0; m<NbrMCsimulation; m++)
      {
        pnl_mat_get_row(ptLib_current->libor, LiborPathsM
  atrix, time_index + m*Nsteps);
        payoff = Swaption_Payoff_Discounted(ptLib_
  current, ptSwpt_current, p, flag_numeraire);
```

```
          // If the payoff is null, the OptimalPayoff doesn
    t change.
          if (payoff>0)
            {
              for (j=0; j<Nfac; j++)
                {
                  VariablesExplicatives[j] = MGET(BrownianM
    atrixPaths, time_index-1 + m*Nsteps, j);
                }

              regressed_value = pnl_basis_eval(basis,LS_Reg
    ressionCoeffVect, VariablesExplicatives);

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

  // The price at date 0 is the conditional expectation of
    OptimalPayoff, ie it's empirical mean.
  *LS_LowerPrice = pnl_vect_sum(OptimalPayoff)/NbrMCsimulat
    ion;

  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 Br
```

```
    oadie algorithm.
 * @param SwaptionPriceUpper upper bound for the price on
    exit.
 * @param NbrMCsimulationDual number of outer simulation
    in Andersen and Broadie algorithm.
 * @param NbrMCsimulationDualInternal  number of inner simu
    lation in Andersen and Broadie algorithm.
 * @param NbrMCsimulationPrimal number of simulation in Lon
    gstaff-Schwartz algorithm.
 */
static void AndersenBroadie(double *SwaptionPriceUpper,
    double Nominal, long NbrMCsimulationDual, long NbrMCsimulationD
    ualInternal, long NbrMCsimulationPrimal, NumFunc_1 *p,
    Libor *ptLib, Swaption *ptBermSwpt, Volatility *ptVol, int     generator, in
    flag_numeraire)
{
  int j, m, m_i, N, k, Nfac, alpha, beta, Nsteps, save_all_
    paths, save_brownian;
  int NbrExerciseDates, start_index, end_index, nbr_var_ex
    plicatives, ExerciceOrContinuation;
  double t, tenor, payoff, payoff_inner, numeraire_0,
    numeraire_i, ContinuationValue, LowerPriceOld, LowerPrice, Low
    erPrice_0, CondExpec_inner=0.;
  double DoobMeyerMartingale, MaxVariable, Delta_0;
  double *VariablesExplicatives;


  PnlMat *LiborPathsMatrix, *BrownianMatrixPaths;
  PnlMat *LiborPathsMatrix_inner, *BrownianMatrixPaths_inn
    er;
  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 swaption maturity
beta  = (int)(ptBermSwpt->swapMaturity/tenor); // T(beta)
   is the swap maturity
NbrExerciseDates = beta-alpha;

nbr_var_explicatives = Nfac;
VariablesExplicatives = malloc(nbr_var_explicatives*size
  of(double));
basis = pnl_basis_create(basis_name, DimApprox, nbr_var_e
  xplicatives);

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 da
  te de la bermudeene. sa maturite reste fixe.
mallocSwaption(&ptSwpt_current, ptBermSwpt->swaptionMatu
  rity, ptBermSwpt->swapMaturity, 0.0, ptBermSwpt->strike, ten
  or);
mallocSwaption(&ptSwpt_current_eur, ptBermSwpt->swaptionM
  aturity, ptBermSwpt->swapMaturity, 0.0, ptBermSwpt->strike,
  tenor);

// calcul de la borne inf du prix et des coefficients de
  regression.
MC_BermSwaption_LongstaffSchwartz(&LowerPrice_0, NbrMCs
  imulationPrimal, p, ptLib, ptBermSwpt, ptVol, generator,
  basis_name, DimApprox, NbrStepPerTenor, flag_numeraire, LS_
  RegressionCoeffMat);
```

```
Delta_0 = 0;

save_brownian = 1;  // save_brownian = 1, we store the
  value brownian motion used in the simulation.
save_all_paths = 1; // save_all_paths = 0, we store only
  the simulated value of libors at the end, ie T(end_index).

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

Sim_Libor_Glasserman(start_index, end_index, ptLib, pt    Vol, generator, NbrM
  all_paths, LiborPathsMatrix, save_brownian, BrownianMatrix
  Paths, flag_numeraire);

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

    pnl_mat_get_row(ptLib_current->libor, LiborPathsMatr
  ix, start_index + m*Nsteps);
    ptSwpt_current->swaptionMaturity = ptBermSwpt->swapt
  ionMaturity; // 1ere opportunite d'exercice
    payoff = Swaption_Payoff_Discounted(ptLib_current, pt
  Swpt_current, p, flag_numeraire);

    pnl_mat_get_row(LS_RegressionCoeffVect, LS_Regression
  CoeffMat, 0);

    for (j=0; j<Nfac; j++)
      {
        VariablesExplicatives[j] = MGET(BrownianMatrix
  Paths, start_index-1 + m*Nsteps, j);
      }

    ContinuationValue = pnl_basis_eval(basis,LS_Regressi
  onCoeffVect, VariablesExplicatives);

    LowerPrice = MAX(ContinuationValue, payoff); // Prix
  d'apres Longstaff/Schwartz a l'instant ptBermSwpt->swaptio
```

```
nMaturity.
  DoobMeyerMartingale = LowerPrice; // initialisation
de la martingale utilisee dans la borne sup.
  LowerPriceOld = LowerPrice;

  MaxVariable = payoff-DoobMeyerMartingale; // initia
lisation de la variable duale dont on calculera l'esparance.
  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(t, p, ptLib_current, ptVol, ptSwpt_
current_eur);

      //ExerciceOrContinuation = (payoff>ContinuationV
alue && payoff>eur_swaption_price);
      ExerciceOrContinuation = payoff>ContinuationValu
e;

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

      // Si ExerciceOrContinuation=Exercice, on calcul
e l'esperance conditionnelle du prix LS.
      if (ExerciceOrContinuation)
        {
          Sim_Libor_Glasserman(start_index, end_index,
ptLib_current, ptVol, generator, NbrMCsimulationDualIntern
al, NbrStepPerTenor, 0, LiborPathsMatrix_inner, 1, BrownianM
atrixPaths_inner, flag_numeraire);

          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_current, p, flag_numeraire);

              for (j=0; j<Nfac; j++)
                {
                  VariablesExplicatives[j] = MGET(Brow
nianMatrixPaths,  start_index-1 + m*Nsteps, j) + MGET(Brow
nianMatrixPaths_inner, m_i, j);
                }

              ContinuationValue = pnl_basis_eval(basis,
LS_RegressionCoeffVect, VariablesExplicatives);

              CondExpec_inner += MAX(payoff_inner,
ContinuationValue);
            }

          CondExpec_inner /= (double) NbrMCsimulationD
ualInternal;

        }

      // calcul du prix LS a la date T_(start_index+k+1
)
      pnl_mat_get_row(ptLib_current->libor, LiborPathsM
atrix, end_index + m*Nsteps);
      payoff = Swaption_Payoff_Discounted(ptLib_
current, ptSwpt_current, p, flag_numeraire);

      for (j=0; j<Nfac; j++)
        {
          VariablesExplicatives[j] = MGET(BrownianMatr
ixPaths, start_index + m*Nsteps, j);
        }

      ContinuationValue = pnl_basis_eval(basis,LS_Reg
ressionCoeffVect, VariablesExplicatives);
```

```
      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_inner;

        }
      else
        {
          DoobMeyerMartingale = DoobMeyerMartingale +
LowerPrice - LowerPriceOld;
        }

      MaxVariable = MAX(MaxVariable, payoff-DoobMeyerM
artingale);
      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_swa
ption_ap_rebonato(t, p, ptLib_current, ptVol, ptSwpt_
current_eur);

  //ExerciceOrContinuation = (payoff>ContinuationValue
&& payoff>eur_swaption_price);
  ExerciceOrContinuation = payoff>ContinuationValue;
```

```
  ptSwpt_current->swaptionMaturity += tenor; // derni
ere opportunite d'exercice = SwapMat-tenor

  if (ExerciceOrContinuation) //Si ExerciceOrContinua
tion==Exercice, on calcule l'esperance conditionnelle
    {
      Sim_Libor_Glasserman(start_index, end_index, pt
Lib_current, ptVol, generator, NbrMCsimulationDualInternal,
NbrStepPerTenor, 0, LiborPathsMatrix_inner, 0, BrownianMatr
ixPaths_inner, flag_numeraire);

      CondExpec_inner = 0;
      for (m_i=0; m_i<NbrMCsimulationDualInternal; m_i+
+)
        {
          pnl_mat_get_row(ptLib_inner->libor, LiborPath
sMatrix_inner, m_i);
          payoff_inner = Swaption_Payoff_Discounted(pt
Lib_inner, ptSwpt_current, p, flag_numeraire);

          CondExpec_inner += payoff_inner; // le prix
a la derniere opportunite d'exercice est = payoff
        }

      CondExpec_inner /= NbrMCsimulationDualInternal;
    }

  pnl_mat_get_row(ptLib_current->libor, LiborPathsMatr
ix, end_index + m*Nsteps);
  payoff = Swaption_Payoff_Discounted(ptLib_current, pt
Swpt_current, p, flag_numeraire);

  LowerPrice = payoff;

  if (ExerciceOrContinuation)
    {
      DoobMeyerMartingale = DoobMeyerMartingale + Low
erPrice - CondExpec_inner;
    }
  else
    {
```

```
        DoobMeyerMartingale = DoobMeyerMartingale + Low
    erPrice - LowerPriceOld;
        }

      MaxVariable = MAX(MaxVariable, payoff-DoobMeyerMartin
    gale);

      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_factors, double swap_maturity,
    double swaption_maturity,  double Nominal, double swaption_stri
    ke, double tenor, long NbrMCsimulationPrimal, long NbrMCs
    imulationDual, long NbrMCsimulationDualInternal, int     generator, int basi
    flag_numeraire, double *swaption_price_upper)
{
  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_matu
    rity, 0.0, swaption_strike, tenor);

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

  AndersenBroadie(swaption_price_upper, Nominal, NbrMCsimu
    lationDual, NbrMCsimulationDualInternal, NbrMCsimulationPrim
    al, p, ptLib, ptBermSwpt, ptVol, generator, basis_name, Dim
    Approx, NbrStepPerTenor, flag_numeraire);

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

  return init_mc;
}

int CALC(MC_AndersenBroadie_BermudanSwaption)(void *Opt,
    void *Mod, PricingMethod *Met)
{
  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_PDOUB
    LE,
                                 ptMod->NbFactors.Val.V_
    ENUM.value,
```

```
                                ptOpt->BMaturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,
                                ptOpt->OMaturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,
                                ptOpt->Nominal.Val.V_PDOUB
    LE,
                                ptOpt->FixedRate.Val.V_PDO
    UBLE,
                                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.val
    ue,
                                Met->Par[4].Val.V_ENUM.val
    ue,
                                Met->Par[5].Val.V_INT,
                                Met->Par[6].Val.V_INT,
                                Met->Par[7].Val.V_ENUM.val
    ue,
                                &(Met->Res[0].Val.V_
    DOUBLE));
}

static int CHK_OPT(MC_AndersenBroadie_BermudanSwaption)(voi
    d *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=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)
};
```

References