Help

```
#include <stdlib.h>
#include "bs1d_std.h"
#include "enums.h"
#include "pnl/pnl_basis.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(MC_MLSM_WANGCAFLISCH)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_MLSM_WANGCAFLISCH)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


void BS_PathsSimulation(PnlMat *SpotPaths, double S0, double Maturity, double r,
{
  int i, m;
  double time_step, a, b;

  pnl_mat_resize(SpotPaths, NbrExerciseDates, NbrMCsimulation);

  time_step = Maturity / (double)(NbrExerciseDates - 1);

  a = (r - divid - SQR(sigma) / 2.) * time_step;
  b = sigma * sqrt(time_step);

  for (m = 0; m < NbrMCsimulation; m++)
    {
      MLET(SpotPaths, 0, m) = S0;
      for (i = 1; i < NbrExerciseDates; i++)
        {
          MLET(SpotPaths, i, m) = MLET(SpotPaths, i - 1, m) * exp(a + b * pnl_ra
        }
    }
}
/** Price of american put/call option using Longstaff-Schwartz algorithm **/
```

```
/** Heston model is simulated using the method proposed by Alfonsi **/
// Exercice dates are : T(0), T(1), ..., T(NbrExerciseDates-1).
// with T(0)=0 and T(NbrExerciseDates-1)=Maturity.
static int MC_MLSM_WANGCAFLISCH(NumFunc_1 *p, double S0, double Maturity, double
{
  int j, m, m_in_money, nbr_var_explicatives, init_mc;
  double a, b, S_init, continuation_value, discounted_payoff, S_t;
  double discount_step, discount, time_step, exercise_date, alpha;
  double *VariablesExplicatives;

  PnlMat  *OneSpotPaths, *SpotPaths, *ExplicativeVariables;
  PnlVect *OptimalPayoff, *RegressionCoeffVect;
  PnlVect *VectToRegress;
  PnlBasis *basis;

  init_mc = pnl_rand_init(generator, NbrExerciseDates * NbrStepPerPeriod, NbrMCs
  if (init_mc != OK) return init_mc;

  alpha = 0.05;
  nbr_var_explicatives = 1;

  basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives);

  VariablesExplicatives = malloc(nbr_var_explicatives * sizeof(double));

  ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nbr_var_explicatives);
  OptimalPayoff = pnl_vect_create(NbrMCsimulation);  // Payoff if following opti
  VectToRegress = pnl_vect_create(NbrMCsimulation);

  RegressionCoeffVect = pnl_vect_create(0); // Regression coefficient.
  SpotPaths = pnl_mat_create(NbrExerciseDates, NbrMCsimulation); // Matrix of th
  OneSpotPaths = pnl_mat_new();

  time_step = Maturity / (double)(NbrExerciseDates - 1);
  discount_step = exp(-r * time_step);
  discount = exp(-r * Maturity);

  b = sigma * sqrt(Maturity * alpha);
  a = SQR(b) / 2.;
  // Simulation of the whole paths
  for (m = 0; m < NbrMCsimulation; m++)
```

```
    {
      //S_init = S0*exp(-a + b*pnl_rand_normal(generator));
      S_init = S0 * exp(-a + b * pnl_inv_cdfnor((double)(m + 1) / (double)(NbrMC

      BS_PathsSimulation(OneSpotPaths, S_init, Maturity, r, divid, sigma, 1, Nbr

      for (j = 0; j < NbrExerciseDates; j++)
        {
          MLET(SpotPaths, j, m) = MGET(OneSpotPaths, j, 0);
        }
    }

  // At maturity, the price of the option = discounted_payoff
  exercise_date = Maturity;
  for (m = 0; m < NbrMCsimulation; m++)
    {
      S_t = MGET(SpotPaths, NbrExerciseDates - 1, m); // Simulated value of the
      LET(OptimalPayoff, m) = discount * (p->Compute)(p->Par, S_t) / S0; // Disc
    }

  for (j = NbrExerciseDates - 2; j >= 0; j--)
    {
      /** Least square fitting **/
      exercise_date -= time_step;
      discount /= discount_step;

      m_in_money = 0;
      pnl_mat_resize(ExplicativeVariables, NbrMCsimulation, nbr_var_explicatives
      pnl_vect_resize(VectToRegress, NbrMCsimulation);
      for (m = 0; m < NbrMCsimulation; m++)
        {
          S_t = MGET(SpotPaths, j, m); // Simulated value of the spot at t=exerc

          discounted_payoff = discount * (p->Compute)(p->Par, S_t) / S0;

          if (discounted_payoff > 0)
            {
              MLET(ExplicativeVariables, m_in_money, 0) = S_t / S0;

              LET(VectToRegress, m_in_money) = GET(OptimalPayoff, m);
              m_in_money++;
```

```
          }
      }
    pnl_mat_resize(ExplicativeVariables, m_in_money, nbr_var_explicatives);
    pnl_vect_resize(VectToRegress, m_in_money);

    pnl_basis_fit_ls(basis, RegressionCoeffVect, ExplicativeVariables, VectToR

    /** Dynamical programming equation **/
    for (m = 0; m < NbrMCsimulation; m++)
      {
        S_t = MGET(SpotPaths, j, m);
        discounted_payoff = discount * (p->Compute)(p->Par, S_t) / S0;

        if (discounted_payoff > 0.) // If the payoff is null, the OptimalPayof
          {
            VariablesExplicatives[0] = S_t / S0;

            continuation_value = pnl_basis_eval(basis, RegressionCoeffVect, Va

            if (discounted_payoff > continuation_value)
              {
                LET(OptimalPayoff, m) = discounted_payoff;
              }
          }
      }
  }

pnl_mat_resize(ExplicativeVariables, NbrMCsimulation, nbr_var_explicatives);
pnl_vect_resize(VectToRegress, NbrMCsimulation);
for (m = 0; m < NbrMCsimulation; m++)
  {
    S_t = MGET(SpotPaths, 0, m);
    MLET(ExplicativeVariables, m, 0) = S_t / S0;
    LET(VectToRegress, m) = GET(OptimalPayoff, m);
  }
pnl_basis_fit_ls(basis, RegressionCoeffVect, ExplicativeVariables, VectToRegre


VariablesExplicatives[0] = 1.;

*ptPriceAm = S0 * pnl_basis_eval(basis, RegressionCoeffVect, VariablesExplicat
```

```
  *ptDeltaAm = pnl_basis_eval_D(basis, RegressionCoeffVect, VariablesExplicative

  free(VariablesExplicatives);
  pnl_basis_free(&basis);
  pnl_mat_free(&SpotPaths);
  pnl_mat_free(&ExplicativeVariables);
  pnl_mat_free(&OneSpotPaths);

  pnl_vect_free(&OptimalPayoff);
  pnl_vect_free(&RegressionCoeffVect);
  pnl_vect_free(&VectToRegress);

  return OK;
}

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

  double r, divid;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
  Met->Par[1].Val.V_INT = MAX(2, Met->Par[1].Val.V_INT); // At least two exercis


  return MC_MLSM_WANGCAFLISCH(ptOpt->PayOff.Val.V_NUMFUNC_1,
                              ptMod->S0.Val.V_PDOUBLE,
                              ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                              r,
                              divid,
                              ptMod->Sigma.Val.V_PDOUBLE,
                              Met->Par[0].Val.V_LONG,
                              Met->Par[1].Val.V_INT,
                              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),
                              &(Met->Res[1].Val.V_DOUBLE));
```

```
}

static int CHK_OPT(MC_MLSM_WANGCAFLISCH)(void *Opt, void *Mod)
{
  Option *ptOpt = (Option *)Opt;
  TYPEOPT *opt = (TYPEOPT *)(ptOpt->TypeOpt);

  if ((opt->EuOrAm).Val.V_BOOL == AMER)
    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_INT = 20;
      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 = &PremiaEnumBasis;
      Met->Par[5].Val.V_INT = 10;
    }

  return OK;
}


PricingMethod MET(MC_MLSM_WANGCAFLISCH) =
{
  "MC_MLSM_WangCaflisch",
  {
    {"N Simulations", LONG, {100}, ALLOW},
    {"N Exercise Dates", INT, {100}, ALLOW},
    {"N Steps per Period", INT, {100}, ALLOW},
```

```
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Basis", ENUM, {100}, ALLOW},
    {"Dimension Approximation", INT, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_MLSM_WANGCAFLISCH),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Delta", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_MLSM_WANGCAFLISCH),
  CHK_ok,
  MET(Init)
};
```