Help

```
#include <stdlib.h>
#include  "
href../../../mod/doublehes1d/doublehes1d_std/doublehes1d_std_h_src.pdfhes1d_std.
#include "pnl/pnl_basis.h"
#include  "
href../../../common/math/alfonsi_h_src.pdfmath/alfonsi.h"
#include "
href../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2011+2) //The "#els
static int CHK_OPT(MC_AM_Alfonsi_Iterative)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_AM_Alfonsi_Iterative)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

#define nbr_var_explicatives_LoSc 2


/** The principle the iterative algorithm is start with a first exercise strateg
In this code we consider two different initial strategies:

1) The strategy given by the Longstaff-Schwartz algorithm
2) The strategy that exercises when the payoff is greater than the prices of all
**/

/** Estimate the exercise strategy given by the Longstaff-Schwartz algorithm **/
/* Exercice dates are : T(0), T(1), ..., T(NbrExerciseDates-1), with T(0)=0 and
The exercise strategy is: at time T(i) we exercise if DiscountedPayoff(T(i))>=Co
where Continuationvalue(T(i)) is given by regession. The regession coefficient a
static int MC_Am_Alfonsi_LoSc(NumFunc_1 *p, double S0, double Maturity, double r
{
  int j, m;
  double regressed_value, discounted_payoff, S_t, V_t, discount, discount_step,
  double *VariablesExplicatives;
```

```
int NbrMCsimulation = SpotPaths->n, NbrExerciseDates = SpotPaths->m;

PnlMat *ExplicativeVariables;
PnlVect *DiscountedOptimalPayoff, *RegressionCoeffVect;
PnlBasis *basis;

pnl_mat_resize(RegressionCoeffMat, NbrExerciseDates, DimApprox);
pnl_mat_set_double(RegressionCoeffMat, 0.);

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

/* We store Spot and Variance*/

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives_LoSc);

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

ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nbr_var_explicatives_Lo
DiscountedOptimalPayoff = pnl_vect_create(NbrMCsimulation); // Continuation Va

RegressionCoeffVect = pnl_vect_create(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);
    LET(DiscountedOptimalPayoff, m) = discount * (p->Compute)(p->Par, S_t); //
  }

// Backward iterations
for (j = NbrExerciseDates - 2; j >= 1; j--)
  {
    /** Least square fitting **/
    exercise_date -= time_step;
    discount /= discount_step;

    for (m = 0; m < NbrMCsimulation; m++)
      {
```

```
        V_t = MGET(VarPaths, j, m); // Simulated value of the variance
        S_t = MGET(SpotPaths, j, m); // Simulated value of the spot

        MLET(ExplicativeVariables, m, 0) = S_t / S0;
        MLET(ExplicativeVariables, m, 1) = V_t / V0;
      }

    pnl_basis_fit_ls(basis, RegressionCoeffVect, ExplicativeVariables, Discoun

    // Save regression coefficients in RegressionCoeffMat.
    pnl_mat_set_row(RegressionCoeffMat, RegressionCoeffVect, j);

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

        if (discounted_payoff > 0) // If the discounted_payoff is null, the Op
          {
            VariablesExplicatives[0] = S_t / S0;
            VariablesExplicatives[1] = V_t / V0;

            regressed_value = pnl_basis_eval(basis, RegressionCoeffVect, Varia

            if (discounted_payoff > regressed_value)
              {
                LET(DiscountedOptimalPayoff, m) = discounted_payoff;
              }
          }
      }
  }

// At initial date, no need for regression, cond.expectation is just a plain e
ContinuationValue_0 = pnl_vect_sum(DiscountedOptimalPayoff) / NbrMCsimulation;

MLET(RegressionCoeffMat, 0, 0) = ContinuationValue_0;

free(VariablesExplicatives);
pnl_basis_free(&basis);
```

```
  pnl_mat_free(&ExplicativeVariables);

  pnl_vect_free(&DiscountedOptimalPayoff);
  pnl_vect_free(&RegressionCoeffVect);

  return OK;
}


// Initial strategy given by Longstaff-Schwartz algorithm
double LimInitialStrategy_LoSc(NumFunc_1 *p, int i, int NbrExerciseDates, double
{
  double result, *VariablesExplicatives;

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

  VariablesExplicatives[0] = S_i / S0;
  VariablesExplicatives[1] = V_i / V0;

  result = MAX(0., pnl_basis_eval(basis, RegCoeffVect_LoSc, VariablesExplicative
  free(VariablesExplicatives);

  return result;
}

// With this initial strategy, we exercise if the payoff is greater than the max
double LimInitialStrategy_EuOpt(NumFunc_1 *p, int i, int NbrExerciseDates, doubl
{
  double Q_0_tilde_i, european_price = 0., european_delta = 0.;
  int q;

  if (i == NbrExerciseDates - 1) return 0.;
  else
    {
      Q_0_tilde_i =  discount_i * (p->Compute)(p->Par, S_i);

      for (q = i + 1; q < NbrExerciseDates; q++)
        {
          ApAntonelliScarlattiHeston(S_i, p, (q - i)*time_step, r, divid, V_i, k
          Q_0_tilde_i = MAX(Q_0_tilde_i, discount_i * european_price);
        }
```

```
      return Q_0_tilde_i;
    }
}

/* Initial Strategy that will be improved with iterative algorithm.
This initial strategy is defined by:
We exercise at t(i) if DiscountedPayoff(t(i)) is greater than LimInitialStrategy
double LimInitialStrategy(NumFunc_1 *p, int i, int NbrExerciseDates, double time
                          double r, double divid, double V_i, double V0, double
                          PnlBasis *basis, PnlVect *RegCoeffVect_LoSc, int flag_
{
  double Q_0_tilde_i = 0.;

  if (flag_InitStrategy == 1)
    {
      Q_0_tilde_i = LimInitialStrategy_EuOpt(p, i, NbrExerciseDates, time_step,
    }

  else if (flag_InitStrategy == 2)
    {
      Q_0_tilde_i = LimInitialStrategy_LoSc(p, i, NbrExerciseDates, time_step, S
    }

  return Q_0_tilde_i;
}

/** Price of american put/call option using Iterative algorithm by Kolodko et al
/** 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_Am_Alfonsi_Iterative(NumFunc_1 *p, double S0, double Maturity, dou
{
  int i, j, m, q, init_mc;
  int flag_SpotPaths, flag_VarPaths, flag_AveragePaths;
  double discounted_payoff_j, discounted_payoff_i = 0., discount_step, time_step
  double S_j, V_j, Q_0_tilde_i = 0., exercise_date, S_i, V_i, regressed_value;
  double *VariablesExplicatives;

  PnlVect *discount, *RegCoeffVect_LoSc, *RegCoeffVect_Iter, *DiscountedOptimalP
  PnlMat *SpotPaths, *VarPaths, *AveragePaths,
```

```
       *RegCoeffMat_LoSc, *RegCoeffMat_Iter, *ExplicativeVariables, *CondExp;

PnlMatInt *ExerciseDecision;

PnlBasis *basis;

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

/* We store Spot and Variance*/
flag_SpotPaths = 1;
flag_VarPaths  = 1;
flag_AveragePaths = 0;

SpotPaths = pnl_mat_new(); // Matrix of the whole trajectories of the spot
VarPaths = pnl_mat_new(); // Matrix of the whole trajectories of the variance
AveragePaths = pnl_mat_new(); // This variable wont be used
RegCoeffVect_LoSc = pnl_vect_new(); // Regression coefficients given by Longst
RegCoeffMat_LoSc = pnl_mat_new(); // All Regression coefficients given by Long
RegCoeffVect_Iter = pnl_vect_new(); // Regression coefficients in the Iterativ
RegCoeffMat_Iter = pnl_mat_create(NbrExerciseDates, DimApprox); // All Regress
DiscountedOptimalPayoff = pnl_vect_create(NbrMCsimulation); // Value of the Di
VectToReg = pnl_vect_create(NbrMCsimulation);
ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nbr_var_explicatives_Lo

//ExerciseDecision[i,m]=0 the strategy says "No exercise" at time t(i) for sim
//ExerciseDecision[i,m]=1 the strategy says "Exercise" at time t(i) for simula
ExerciseDecision = pnl_mat_int_create(NbrExerciseDates, NbrMCsimulation);
pnl_mat_int_set_int(ExerciseDecision, 0);

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

CondExp = pnl_mat_create(NbrExerciseDates, NbrMCsimulation);
discount = pnl_vect_create(NbrExerciseDates);

time_step = Maturity / (double)(NbrExerciseDates - 1);
discount_step = exp(-r * time_step);
LET(discount, 0) = 1.;
for (i = 1; i < NbrExerciseDates; i++) LET(discount, i) = discount_step * GET(

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives_LoSc);
```

```
// Simulation of the whole paths
HestonSimulation_Alfonsi(flag_SpotPaths, SpotPaths, flag_VarPaths, VarPaths, f

// If flag_InitStrategy=2, the first strategy will the Longstaff-Schwartz one
if (flag_InitStrategy == 2) MC_Am_Alfonsi_LoSc(p, S0, Maturity, r, divid, V0,


///* Here we compute the dates where the initial strategy says to exercise the
//ExerciseDecision[i,m]=0 the strategy says "No exercise" at time t(i) for sim
//ExerciseDecision[i,m]=1 the strategy says "Exercise" at time t(i) for simula
i = 0; // t=0
S_i = S0;
V_i = V0;
discounted_payoff_i = GET(discount, i) * (p->Compute)(p->Par, S_i);
if (discounted_payoff_i > 0)
  {
    if (flag_InitStrategy == 2) pnl_mat_get_row(RegCoeffVect_LoSc, RegCoeffMat

    Q_0_tilde_i = LimInitialStrategy(p, i, NbrExerciseDates, time_step, S_i, S

    if (discounted_payoff_i >= Q_0_tilde_i)
      {
        for (m = 0; m < NbrMCsimulation; m++)
          pnl_mat_int_set(ExerciseDecision, i, m, 1);
      }
  }

for (i = 1; i < NbrExerciseDates - 1; i++)
  {
    if (flag_InitStrategy == 2)
      pnl_mat_get_row(RegCoeffVect_LoSc, RegCoeffMat_LoSc, i);

    for (m = 0; m < NbrMCsimulation; m++)
      {
        S_i = MGET(SpotPaths, i, m);
        V_i = MGET(VarPaths, i, m);

        discounted_payoff_i = GET(discount, i) * (p->Compute)(p->Par, S_i);

        if (discounted_payoff_i > 0)
```

```
          {
            Q_0_tilde_i = LimInitialStrategy(p, i, NbrExerciseDates, time_step

            if (discounted_payoff_i >= Q_0_tilde_i)
              pnl_mat_int_set(ExerciseDecision, i, m, 1);
          }
      }
  }

for (m = 0; m < NbrMCsimulation; m++)
  pnl_mat_int_set(ExerciseDecision, NbrExerciseDates - 1, m, 1);


///* We compute the option price with the initial exercise strategy
*ptPriceAm_1stStrategy = 0.;
for (m = 0; m < NbrMCsimulation; m++)
  {
    i = -1;
    do
      {
        i++;
      }
    while (pnl_mat_int_get(ExerciseDecision, i, m) == 0);

    S_i = MGET(SpotPaths, i, m);
    V_i = MGET(VarPaths, i, m);
    discounted_payoff_i = GET(discount, i) * (p->Compute)(p->Par, S_i);
    *ptPriceAm_1stStrategy += discounted_payoff_i;
  }
*ptPriceAm_1stStrategy /= (double) NbrMCsimulation; // Price with initial exer


///* Now we compute the option price following the improved exercise strategy
// At maturity, the price of the option = discounted_payoff
// DiscountedOptimalPayoff will contain the disc.payoff following the improved
exercise_date = Maturity;
for (m = 0; m < NbrMCsimulation; m++)
  {
    S_j = MGET(SpotPaths, NbrExerciseDates - 1, m);
    LET(DiscountedOptimalPayoff, m) = GET(discount, NbrExerciseDates - 1) * (p
  }
```

```
for (j = NbrExerciseDates - 2; j >= 1; j--)
  {
    exercise_date -= time_step;

    for (m = 0; m < NbrMCsimulation; m++)
      {
        S_j = MGET(SpotPaths, j, m);
        V_j = MGET(VarPaths, j, m);

        MLET(ExplicativeVariables, m, 0) = S_j / S0;
        MLET(ExplicativeVariables, m, 1) = V_j / V0;

        i = j - 1;
        for (q = j; q < MIN(j + WindowPar + 1, NbrExerciseDates); q++)
          {
            if (i < q)
              {

                do
                  {
                    i++;
                  }
                while (pnl_mat_int_get(ExerciseDecision, i, m) == 0);

                S_i = MGET(SpotPaths, i, m);
                V_i = MGET(VarPaths, i, m);
                discounted_payoff_i = GET(discount, i) * (p->Compute)(p->Par,

                MLET(CondExp, q, m) = discounted_payoff_i;
              }
            else MLET(CondExp, q, m) = discounted_payoff_i;
          }
      }


    for (q = j; q < MIN(j + WindowPar + 1, NbrExerciseDates); q++)
      {
        pnl_mat_get_row(VectToReg, CondExp, q);

        pnl_basis_fit_ls(basis, RegCoeffVect_Iter, ExplicativeVariables, VectT
```

```
        pnl_mat_set_row(RegCoeffMat_Iter, RegCoeffVect_Iter, q); // Save regre
      }

    for (m = 0; m < NbrMCsimulation; m++)
      {
        S_j = MGET(SpotPaths, j, m);
        V_j = MGET(VarPaths, j, m);
        discounted_payoff_j = GET(discount, j) * (p->Compute)(p->Par, S_j);

        if (discounted_payoff_j > 0) // If the discounted_payoff is null, the
          {
            VariablesExplicatives[0] = S_j / S0;
            VariablesExplicatives[1] = V_j / V0;

            regressed_value = 0.;
            for (q = j; q < MIN(j + WindowPar + 1, NbrExerciseDates); q++)
              {
                pnl_mat_get_row(RegCoeffVect_Iter, RegCoeffMat_Iter, q);
                regressed_value = MAX(regressed_value, pnl_basis_eval(basis, R
              }

            if (discounted_payoff_j >= regressed_value)
              {
                LET(DiscountedOptimalPayoff, m) = discounted_payoff_j;
              }
          }
      }
  }


/* Price estimator */
*ptPriceAm_2ndStrategy = MAX((p->Compute)(p->Par, S0), pnl_vect_sum(Discounted

free(VariablesExplicatives);

pnl_basis_free(&basis);

pnl_vect_free(&VectToReg);
pnl_vect_free(&discount);
pnl_vect_free(&RegCoeffVect_LoSc);
```

```
  pnl_vect_free(&RegCoeffVect_Iter);
  pnl_vect_free(&DiscountedOptimalPayoff);

  pnl_mat_free(&ExplicativeVariables);
  pnl_mat_free(&CondExp);
  pnl_mat_free(&SpotPaths);
  pnl_mat_free(&VarPaths);
  pnl_mat_free(&AveragePaths);
  pnl_mat_free(&RegCoeffMat_LoSc);
  pnl_mat_free(&RegCoeffMat_Iter);
  pnl_mat_int_free(&ExerciseDecision);

  return OK;
}

int CALC(MC_AM_Alfonsi_Iterative)(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[2].Val.V_INT = MAX(2, Met->Par[2].Val.V_INT); // At least two exercis


  return MC_Am_Alfonsi_Iterative(ptOpt->PayOff.Val.V_NUMFUNC_1,
                                 ptMod->S0.Val.V_PDOUBLE,
                                 ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DAT
                                 r,
                                 divid,
                                 ptMod->Sigma0.Val.V_PDOUBLE,
                                 ptMod->MeanReversion.Val.V_PDOUBLE,
                                 ptMod->LongRunVariance.Val.V_PDOUBLE,
                                 ptMod->Sigma.Val.V_PDOUBLE,
                                 ptMod->Rho.Val.V_PDOUBLE,
                                 Met->Par[0].Val.V_ENUM.value,
                                 Met->Par[1].Val.V_LONG,
                                 Met->Par[2].Val.V_INT,
                                 Met->Par[3].Val.V_INT,
```

```
                                        Met->Par[4].Val.V_ENUM.value,
                                        Met->Par[5].Val.V_ENUM.value,
                                        Met->Par[6].Val.V_ENUM.value,
                                        Met->Par[7].Val.V_INT,
                                        Met->Par[8].Val.V_INT,
                                        &(Met->Res[0].Val.V_DOUBLE),
                                        &(Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(MC_AM_Alfonsi_Iterative)(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 PremiaEnumMember InitStrategyMembers[] =
{
  { "Andersen-like strategy", 1 },
  { "Longstaff-Schwartz strategy", 2 },
  { NULL, NULLINT }
};

static DEFINE_ENUM(InitStrategy, InitStrategyMembers);

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;

      Met->Par[0].Val.V_ENUM.value = 1;
      Met->Par[0].Val.V_ENUM.members = &InitStrategy;
      Met->Par[1].Val.V_LONG = 50000;
      Met->Par[2].Val.V_INT = 10;
      Met->Par[3].Val.V_INT = 1;
```

```
      Met->Par[4].Val.V_ENUM.value = 0;
      Met->Par[4].Val.V_ENUM.members = &PremiaEnumRNGs;
      Met->Par[5].Val.V_ENUM.value = 2;
      Met->Par[5].Val.V_ENUM.members = &PremiaEnumCirOrder;
      Met->Par[6].Val.V_ENUM.value = 0;
      Met->Par[6].Val.V_ENUM.members = &PremiaEnumBasis;
      Met->Par[7].Val.V_INT = 10;
      Met->Par[8].Val.V_INT = 5;
    }

  return OK;
}


PricingMethod MET(MC_AM_Alfonsi_Iterative) =
{
  "MC_AM_Alfonsi_Iterative",
  {
    {"Initial Strategy", ENUM, {100}, ALLOW},
    {"N Simulations", LONG, {100}, ALLOW},
    {"N Exercise Dates", INT, {100}, ALLOW},
    {"N Steps per Period", INT, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Cir Order", ENUM, {100}, ALLOW},
    {"Basis", ENUM, {100}, ALLOW},
    {"Dimension Approximation", INT, {100}, ALLOW},
    {"Window Parameter", INT, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_AM_Alfonsi_Iterative),
  { {"Price by 2nd Strategy", DOUBLE, {100}, FORBID},
    {"Price by 1st  Strategy", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_AM_Alfonsi_Iterative),
  CHK_ok,
  MET(Init)
};
```