Help

```
#include <stdlib.h>
#include "hes1d_pad.h"
#include "pnl/pnl_basis.h"
#include "math/alfonsi.h"
#include "math/moments.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(MC_Am_Asian_Alfonsi_LongstaffSchwartz_hes1d)(void *Opt, void
{
  return NONACTIVE;
}
int CALC(MC_Am_Asian_Alfonsi_LongstaffSchwartz_hes1d)(void *Opt, void *Mod, Pric
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/** Functions used in the regression basis in Longstaff-Schwartz algorithm **/
// Approximation formula for a european asian-option under Black-Scholes model.
static int Analytic_KemnaVorst_BlackScholes(double pseudo_stock, double pseudo_s
{
  double b;

  b = r - divid;

  /* Put Case */
  if ((p->Compute) == &Put_OverSpot2)
    {
      *ptprice = 0.;
      *ptdelta = 0.;
    }
  /* Call case */
  else
    {
      /* Case r=d */
      if (b == 0.)
        {
          *ptprice = exp(-r * t) * (pseudo_stock - pseudo_strike);
```

```
          *ptdelta = exp(-r * t) * (1. - time_spent);
        }
      /* Case r <> d */
      else
        {
          *ptprice = exp(-r * t) * (pseudo_stock * (1.0 / (b * t)) * (exp(b * t)
          *ptdelta = exp(-r * t) * ((1 - time_spent) / (b * t) * (exp(b * t) - 1
        }
    }
  return OK;
}

static int Levy_FixedAsian_BlackScholes(double pseudo_stock, double pseudo_strik
{

  double m1, m2, m, v, d1, d2, esp, nd1, nd2;
  double CTtK, PTtK, Dlt, Plt;
  double new_r, new_sigma;

  /*Computation of the first two moments*/
  new_r = (r - divid) * t;
  new_sigma = sigma * sqrt(t);
  m1 = Moments(1, new_r, new_sigma, 1);
  m2 = Moments(2, new_r, new_sigma, 1);

  /*Fit the parameters m,v of lognormal distribution*/
  m = 2.0 * log(m1) - log(m2) / 2.0;
  v = sqrt(fabs(log(m2) - 2.0 * log(m1)));

  /*Adjusted input for Black-Scholes Formula*/
  d1 = (log(pseudo_stock / pseudo_strike) + m + SQR(v)) / v;
  d2 = d1 - v;
  esp = m + SQR(v) / 2.0 - (r - divid) * t;
  nd1 = cdf_nor(d1);
  nd2 = cdf_nor(d2);

  /*  Call Price */
  CTtK = pseudo_stock * exp(-divid * t) * exp(esp) * nd1 - exp(-r * t) * pseudo_

  /*  Put Price from Parity*/
  if (r == divid)
```

```
    PTtK = CTtK + pseudo_strike * exp(-r * t) - pseudo_stock * exp(-r * t);
  else
    PTtK = CTtK + pseudo_strike * exp(-r * t) - pseudo_stock * exp(-r * t) * (ex

  /*Delta for call option*/
  Dlt = exp(esp) * nd1 * exp(-divid * t);

  /*Delta for put option*/
  if (r == divid)
    Plt = Dlt - exp(-r * t);
  else
    Plt = Dlt - exp(-r * t) * (exp((r - divid) * t) - 1.0) / (t * (r - divid));

  /*Price*/
  if ((po->Compute) == &Call_OverSpot2)
    *ptprice = CTtK;
  else
    *ptprice = PTtK;

  /*Delta */
  if ((po->Compute) == &Call_OverSpot2)
    *ptdelta = Dlt;
  else
    *ptdelta = Plt;

  return OK;
}

// Average Starting Date = 0.;
int Ap_FixedAsian_BlackScholes(double Current_Spot, double Current_Avg, double C
{
  int return_value;
  double time_spent, true_strike, pseudo_spot, pseudo_strike;

  true_strike = p->Par[0].Val.V_PDOUBLE;

  time_spent = Current_Date / Maturity;

  pseudo_spot = (1. - time_spent) * Current_Spot;
  pseudo_strike = true_strike - time_spent * Current_Avg;
```

```
  if (pseudo_strike <= 0.)
    return_value = Analytic_KemnaVorst_BlackScholes(pseudo_spot, pseudo_strike,

  else
    return_value = Levy_FixedAsian_BlackScholes(pseudo_spot, pseudo_strike, p, M

  return return_value;

}


/** Lower bound for american option using Longstaff-Schwartz algorithm **/
// Exercice dates are : T(0), T(1), ..., T(NbrExerciseDates-1).
// with T(0)=0 and T(NbrExerciseDates-1)=Maturity.
static int MC_Am_Asian_Alfonsi_LoSc(NumFunc_2  *p, double S0, double Maturity, d
{
  int j, m, nbr_var_explicatives, init_mc;
  int flag_SpotPaths, flag_VarPaths, flag_AveragePaths;
  double continuation_value, discounted_payoff, S_t, V_t, A_t, mean_price, var_p
  double discount_step, discount, step, exercise_date, european_price, european_
  double *VariablesExplicatives;

  PnlMat *SpotPaths, *VarPaths, *AveragePaths, *ExplicativeVariables;
  PnlVect *DiscountedOptimalPayoff, *RegressionCoeffVect;
  PnlBasis *basis;

  european_price = 0.;
  european_delta = 0.;

  /* Value to construct the confidence interval */
  z_alpha = pnl_inv_cdfnor((1. + confidence) / 2.);

  // Time step and discount factor.
  step = Maturity / (double)(NbrExerciseDates - 1);
  discount_step = exp(-r * step);
  discount = exp(-r * Maturity);

  /* We store Spot, Variance and Average*/
  flag_SpotPaths = 1;
  flag_VarPaths  = 1;
  flag_AveragePaths = 1;
```

```
// Number of explicatives variables
nbr_var_explicatives = 2;

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);
DiscountedOptimalPayoff = pnl_vect_create(NbrMCsimulation); // Payoff if follo

RegressionCoeffVect = pnl_vect_create(0); // Regression coefficient.
SpotPaths = pnl_mat_create(0, 0); // Matrix of the whole trajectories of the s
VarPaths = pnl_mat_create(0, 0); // Matrix of the whole trajectories of the va
AveragePaths = pnl_mat_create(0, 0); // Matrix of the whole trajectories of th

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

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

// At maturity, DiscountedOptimalPayoff = discounted_payoff
exercise_date = Maturity;
for (m = 0; m < NbrMCsimulation; m++)
  {
    S_t = MGET(SpotPaths, NbrExerciseDates - 1, m); // Simulated Value of the
    A_t = MGET(AveragePaths, NbrExerciseDates - 1, m); // Simulated Value of t

    LET(DiscountedOptimalPayoff, m) = discount * (p->Compute)(p->Par, S_t, A_t
  }

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

    for (m = 0; m < NbrMCsimulation; m++)
      {
        V_t = MGET(VarPaths, j, m); // Simulated value of the variance at t=ex
```

```
          S_t = MGET(SpotPaths, j, m); // Simulated value of the spot at t=exerc
          A_t = MGET(AveragePaths, j, m); // Simulated value of the average at t

          // Regression basis contains price and delta of european asian option
          // As BS volatility, we take sqrt of expectation of V(Maturity) knowin
          V_mean = theta + (V_t - theta) * exp(-k * (Maturity - exercise_date));
          Ap_FixedAsian_BlackScholes(S_t, A_t, exercise_date, p, Maturity, r, di

          MLET(ExplicativeVariables, m, 0) = discount * european_price / S0;
          MLET(ExplicativeVariables, m, 1) = discount * european_delta * S_t * s
        }

      pnl_basis_fit_ls(basis, RegressionCoeffVect, ExplicativeVariables, Discoun

      /** Dynamical programming equation **/
      for (m = 0; m < NbrMCsimulation; m++)
        {
          V_t = MGET(VarPaths, j, m);
          S_t = MGET(SpotPaths, j, m);
          A_t = MGET(AveragePaths, j, m);

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

          if (discounted_payoff > 0.) // If the payoff is null, the OptimalPayof
            {
              V_mean = theta + (V_t - theta) * exp(-k * (Maturity - exercise_dat
              Ap_FixedAsian_BlackScholes(S_t, A_t, exercise_date, p, Maturity, r

              VariablesExplicatives[0] = discount * european_price / S0;
              VariablesExplicatives[1] = discount * european_delta * S_t * sqrt(

              continuation_value = pnl_basis_eval(basis, RegressionCoeffVect, Va

              if (discounted_payoff > continuation_value)
                {
                  LET(DiscountedOptimalPayoff, m) = discounted_payoff;
                }
            }
        }
    }
  discount /= discount_step;
```

```
  // At initial date, no need for regression, continuation value is just a plain
  continuation_value = pnl_vect_sum(DiscountedOptimalPayoff) / (double)NbrMCsimu
  discounted_payoff = discount * (p->Compute)(p->Par, S0, S0);

  /* Price */
  mean_price = MAX(discounted_payoff, continuation_value);

  /* Sum of squares */
  var_price = SQR(pnl_vect_norm_two(DiscountedOptimalPayoff)) / (double)NbrMCsim
  var_price = MAX(var_price, SQR(discounted_payoff)) - SQR(mean_price);

  /* Price estimator */
  *ptPriceAm = mean_price;
  *ptPriceAmError = sqrt(var_price / ((double)NbrMCsimulation - 1));

  /* Price Confidence Interval */
  *ptInfPriceAm = *ptPriceAm - z_alpha * (*ptPriceAmError);
  *ptSupPriceAm = *ptPriceAm + z_alpha * (*ptPriceAmError);

  free(VariablesExplicatives);

  pnl_basis_free(&basis);
  pnl_mat_free(&VarPaths);
  pnl_mat_free(&AveragePaths);
  pnl_mat_free(&SpotPaths);
  pnl_mat_free(&ExplicativeVariables);

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

  return OK;
}

int CALC(MC_Am_Asian_Alfonsi_LongstaffSchwartz_hes1d)(void *Opt, void *Mod, Pric
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  double T, t_0, T_0;
  double r, divid, time_spent, pseudo_strike, true_strike, true_spot, pseudo_spo
```

```
int return_value;

Met->Par[1].Val.V_INT = MAX(2, Met->Par[1].Val.V_INT); // At least two exercis

r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

T = ptOpt->Maturity.Val.V_DATE;
T_0 = ptMod->T.Val.V_DATE;
t_0 = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;
time_spent = (T_0 - t_0) / (T - t_0);

if (T_0 < t_0)
  {
    Fprintf(TOSCREEN, "T_0 < t_0, untreated case\ n\ n\ n");
    return_value = WRONG;
  }

/* Case t_0 <= T_0 */
else
  {
    true_spot = ptMod->S0.Val.V_PDOUBLE;
    true_strike = (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;
    initial_average = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[4].Val.V_PDOUBLE;

    pseudo_spot = (1. - time_spent) * true_spot;
    pseudo_strike = true_strike - time_spent * initial_average;


    (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE = pseudo_strike;

    return_value = MC_Am_Asian_Alfonsi_LoSc(ptOpt->PayOff.Val.V_NUMFUNC_2,
                                            pseudo_spot,
                                            T - T_0,
                                            r,
                                            divid,
                                            ptMod->Sigma0.Val.V_PDOUBLE,
                                            ptMod->MeanReversion.Val.V_PDOUBLE
                                            ptMod->LongRunVariance.Val.V_PDOUB
                                            ptMod->Sigma.Val.V_PDOUBLE,
                                            ptMod->Rho.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->Par[6].Val.V_PDOUBLE,
                                             Met->Par[7].Val.V_ENUM.value,
                                             &(Met->Res[0].Val.V_DOUBLE),
                                             &(Met->Res[1].Val.V_DOUBLE),
                                             &(Met->Res[2].Val.V_DOUBLE),
                                             &(Met->Res[3].Val.V_DOUBLE));

      (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE = true_strike;
    }
  return return_value;
}

static int CHK_OPT(MC_Am_Asian_Alfonsi_LongstaffSchwartz_hes1d)(void *Opt, void
{
  if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedAmer") == 0) || (strcmp(((Op
    return OK;

  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 = 100000;
      Met->Par[1].Val.V_INT = 10;
      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;
      Met->Par[6].Val.V_DOUBLE = 0.95;
      Met->Par[7].Val.V_ENUM.value = 2;
      Met->Par[7].Val.V_ENUM.members = &PremiaEnumCirOrder;
    }

  return OK;
}


PricingMethod MET(MC_Am_Asian_Alfonsi_LongstaffSchwartz_hes1d) =
{
  "MC_Am_Asian_Alfonsi_LongstaffSchwartz_hes1d",
  {
    {"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},
    {"Confidence Value", DOUBLE, {100}, ALLOW},
    {"Cir Order", ENUM, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_Am_Asian_Alfonsi_LongstaffSchwartz_hes1d),
  {
    {"Price", DOUBLE, {100}, FORBID},
    {"Error Price", DOUBLE, {100}, FORBID},
    {"Inf Price", DOUBLE, {100}, FORBID},
    {"Sup Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_Am_Asian_Alfonsi_LongstaffSchwartz_hes1d),
  CHK_ok,
  MET(Init)
};
```