Help

```
#include  "
href../../../mod/bs1d/bs1d_pad/bs1d_pad_h_src.pdfbs1d_pad.h"
#include "
href../../../common/enums_h_src.pdfenums.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_basis.h"
#include "pnl/pnl_vector.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2013+2) //The "#els
static int CHK_OPT(MC_MovingAverage_BernhartTankovWarin)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_MovingAverage_BernhartTankovWarin)(void *Opt, void *Mod, PricingMeth
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/**
 * Characteristics of a product
 */
typedef struct _Product  Product;
struct _Product
{
  double r; /*!< interest rate */
  double divid; /*!< dividend rate */
  double spot; /*!< spot */
  double T; /*!< maturity */
  double sigma; /*!< volatility */
  int Nd; /* nombre de dates  d'exercices dans la moving window*/
  int Nl; /* nombre de dates de délai dans la moving window */
  int N; /*!< number of exercise dates */
  int Ibasis; /*!< basis index */
  int degree; /*!< degree of the polynomial regression */
  int dim_reg; /*!< dimension of the regression 1 or 2 */
  int M; /*!< number of iterations of the MC procedure */
  NumFunc_2 *pt_NumFunc; /*!< NumFunc for the payoff */
```

```
};

/**
 * Computes the payoff of a call in dimension 1
 *
 * @param St value of the underlying asset at time t
 * @param Xt value of the moving window at time t
 * @param t current time
 * @param P a Product instance
 *
 * @return the payoff
 */
double payoff_func(double St, double Xt, double t, const Product *P)
{
  return exp(-P->r * t) * (*P->pt_NumFunc->Compute)(P->pt_NumFunc->Par, St, Xt);
}

/**
 *
 *
 * @param Alpha (output) coefficients of the decomposition
 * @param Basis basis
 * @param SXt matrix workspace
 * @param S (input) asset trajectories. An array of vector, each
 * entry of the array being an asset path.
 * @param prix vector of the option prices
 * @param M number of trajectories
 * @param P product instance
 * @param n index of the current time (starts from 0)
 */
static void regression(PnlMat *SXt, PnlVect *Alpha, PnlBasis *Basis, PnlMat **SX
{
  int m;

  //SXt matrice de taille M*2. SXt=(S(tn),X(tn))
  pnl_mat_resize(SXt, P->M, 2);
  /*
   * Extract the value of SX on each path at time n
   */
  for (m = 0 ; m < P->M ; m++)
    {
```

```
      MLET(SXt, m, 0) = MGET(SX[m], n, 0);
      MLET(SXt, m, 1) = MGET(SX[m], n, 1);
    }
  pnl_basis_fit_ls(Basis, Alpha, SXt, prix);
}

/**
 * Draw one path of the model and its moving average
 *
 * @param SX (output) contains one path of the model and of the moving window
 * with size P->N. SX matice de taille (N+1)*2
 * built using the gaussian r.v. G. The first Nd values of
 * X are null
 * @param P instance of the product
 * @param rng random number generator
 */
static void asset_MA(PnlMat *SX, const Product *P, PnlRng *rng)
{
  double h, sqrt_h, drift;
  int i, k;
  double tmp;
  MLET(SX, 0, 0) = P->spot;
  for (i = 0; i < P->Nd + P->Nl; i++)
    {
      MLET(SX, i, 1) = 0.0;
    }
  h = P->T / P->N; /* step size between two exercise dates */
  sqrt_h = sqrt(h);
  drift = (P->r - P->divid - P->sigma * P->sigma / 2.) * h;
  for (k = 0 ; k < P->N ; k++)
    {
      MLET(SX, k + 1, 0) = MGET(SX, k, 0) * exp(drift + P->sigma * sqrt_h * pnl_

    }

  for (k = 0, tmp = 0. ; k < P->Nd ; k++)
    {
      tmp += MGET(SX, k, 0);
    }
  MLET(SX, P->Nd + P->Nl - 1, 1) = 1.0 / (P->Nd) * tmp;
  for (k = P->Nd + P->Nl ; k < P->N + 1 ; k++)
    {
```

```
      MLET(SX, k, 1) = MGET(SX, k - 1, 1) + 1.0 / (P->Nd) * (MGET(SX, k - P->Nl,
    }
}

static void scale_domain(PnlBasis *B, const Product *P)
{
  PnlVect *center, *scale;
  center = pnl_vect_create_from_double(P->dim_reg, P->spot);
  scale = pnl_vect_create_from_double(P->dim_reg, P->spot * exp((P->r - P->divid
  pnl_basis_set_reduced(B, center, scale);
  pnl_vect_free(&center);
  pnl_vect_free(&scale);
}

/**
 * Compute an optimal strategy using Longstaff Schwarz approach
 *
 * @param S (input) asset trajectories. An array of vector, each
 * entry of the array being an asset path.
 * @param tau (output) an integer vector, optimal stragey
 * @param M number of trajectories
 * @param P product instance
 */
static void strategie_optimale(PnlVectInt *tau, PnlMat **SX, const Product *P)
{
  int m; /* indice tirage MC */
  int n; /* indice pas de temps */
  PnlBasis *Basis;
  double tn, h;
  PnlVect *Alpha, *prix;
  PnlMat *SXt;

  pnl_vect_int_resize(tau, P->M);
  prix = pnl_vect_create(P->M);
  SXt = pnl_mat_new();
  Basis = pnl_basis_create_from_degree(P->Ibasis, P->degree, P->dim_reg);
  scale_domain(Basis, P);
  Alpha = pnl_vect_create(Basis->nb_func);
  h = P->T / P->N;

  /*
```

```
 * init at time T
 */
pnl_vect_int_resize(tau, P->M);
pnl_vect_int_set_int(tau, P->N);
for (m = 0 ; m < P->M ; m++)
  {
    double ST = MGET(SX[m], P->N, 0);
    double XT = MGET(SX[m], P->N, 1);
    double payoff = payoff_func(ST, XT, P->T, P);
    LET(prix, m) = payoff;
  }
/*
 * Start iterations
 */
for (n = P->N - 1, tn = P->T - h ; n > P->Nd + P->Nl - 1 ; n--, tn -= h)
  {
    regression(SXt, Alpha, Basis, SX, prix, n, P);
    /*
     * update exercise policy
     */
    for (m = 0 ; m < P->M ; m++)
      {
        const double St = MGET(SX[m], n, 0);
        const double Xt = MGET(SX[m], n, 1);
        const double payoff = payoff_func(St, Xt, tn, P);
        PnlVect SXt = pnl_vect_wrap_mat_row(SX[m], n);
        const double Et = pnl_basis_eval_vect(Basis, Alpha, &SXt);
        /*
         * pnl_basis_eval (Basis, Alpha, &St) may be < 0 because of the
         * polynomial interpolation which cannot happen from a
         * theoretical point of view. Hence we must ensure that payoff >
         * 0 before deciding whether to exercise. Exercise cannot happen
         * at a date when payoff == 0
         */
        if (payoff > 0. && Et < payoff)
          {
            pnl_vect_int_set(tau, m, n);
            LET(prix, m) = payoff;
          }
      }
  }
```

```
  pnl_basis_free(&Basis);
  pnl_vect_free(&Alpha);
  pnl_vect_free(&prix);
  pnl_mat_free(&SXt);
}

static int MovingAverage(double spot, NumFunc_2 *pt_NumFunc, double T, double si
                         int window, int delay, int Ndates, int Nsamples,
                         int degree, int Ibasis, int Irng, double *ptprice)
{
  int m;
  double prix, h;
  PnlRng *rng;
  PnlMat **SX;
  PnlVectInt *tau;
  Product P;

  P.r = r;//taux d'intérêt
  P.divid = divid;//taux de dividende
  P.T = T;//maturité
  P.spot = spot;//psot
  P.N = Ndates;//nb de dates de discrétisation
  P.Nd = window; //nb de dates de discrétisation de la fenêtre glissante
  P.Nl = delay; //nb de dates de discrétisation du retard (on a tjs Nl+Nd <= N)
  P.sigma  = sigma;//volatitité
  P.degree = degree;//degré de la base de polynome
  P.Ibasis = Ibasis;
  P.M  = Nsamples;//nb de tirages MC
  P.pt_NumFunc = pt_NumFunc;
  P.dim_reg = 2;
  h = P.T / P.N;
  rng = PnlRngArray[Irng];
  pnl_rng_sseed(rng, 0);


  /*
   * Init
   */
  SX = malloc(P.M * sizeof(PnlMat *));
  for (m = 0 ; m < P.M ; m++)
```

```
    {
      SX[m] = pnl_mat_create(P.N + 1, 2);
      asset_MA(SX[m], &P, rng);
    }
  tau = pnl_vect_int_new();

  strategie_optimale(tau, SX, &P);

  prix = 0.;
  for (m = 0 ; m < P.M ; m++)
    {
      const int tau_m = pnl_vect_int_get(tau, m);
      const double Stau = MGET(SX[m], tau_m, 0);
      const double Xtau = MGET(SX[m], tau_m, 1);
      const double payoff = payoff_func(Stau, Xtau, tau_m * h, &P);
      prix += payoff;
    }
  prix /= P.M;
  *ptprice = prix;


  /* Free memory */
  pnl_vect_int_free(&tau);
  for (m = 0 ; m < P.M ; m++)
    {
      pnl_mat_free(&(SX[m]));
    }
  free(SX);
  return OK;
}

int CALC(MC_MovingAverage_BernhartTankovWarin)(void *Opt, void *Mod, PricingMeth
{
  int return_value;
  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.);
```

```
  return_value = MovingAverage(ptMod->S0.Val.V_PDOUBLE,
                               ptOpt->PayOff.Val.V_NUMFUNC_2,
                               ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                               ptMod->Sigma.Val.V_PDOUBLE,
                               r,
                               divid,
                               (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[1].Val.V_PI
                               (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[2].Val.V_IN
                               (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PI
                               Met->Par[0].Val.V_LONG,
                               Met->Par[3].Val.V_INT,
                               Met->Par[2].Val.V_ENUM.value,
                               Met->Par[1].Val.V_ENUM.value,
                               &(Met->Res[0].Val.V_DOUBLE)
                              );

  return return_value;
}

static int CHK_OPT(MC_MovingAverage_BernhartTankovWarin)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "MovingAverageCallFixedAmer") == 0) ||
      (strcmp(((Option *)Opt)->Name, "MovingAveragePutFixedAmer") == 0) ||
      (strcmp(((Option *)Opt)->Name, "MovingAverageCallFloatingAmer") == 0) ||
      (strcmp(((Option *)Opt)->Name, "MovingAveragePutFloatingAmer") == 0))
    return OK;
  return WRONG;
}

#endif

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_ENUM.value = 0;
      Met->Par[1].Val.V_ENUM.members = &PremiaEnumMCRNGs;
      Met->Par[2].Val.V_ENUM.value = 0;
```

```
      Met->Par[2].Val.V_ENUM.members = &PremiaEnumBasis;
      Met->Par[3].Val.V_INT = 3;
    }
  return OK;
}

PricingMethod MET(MC_MovingAverage_BernhartTankovWarin) =
{
  "MC_MovingAverage_BernhartTankovWarin",
  { {"N iterations", LONG, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Basis", ENUM, {100}, ALLOW},
    {"Approximation degree", INT, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_MovingAverage_BernhartTankovWarin),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_MovingAverage_BernhartTankovWarin),
  CHK_ok,
  MET(Init)
};
```