Help

```
#include <stdlib.h>
#include  "
href../../../mod/doublehes1d/doublehes1d_std/doublehes1d_std_h_src.pdfhes1d_std.
#include "pnl/pnl_matrix.h"
#include "
href../../../common/enums_h_src.pdfenums.h"

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


/* Calculus of the optimal parameter which minimizes the variance by using the C
static void Simul_theta(int flag_call, int type_generator, int Strike, int step_
{
  int RM = 100000; /* number of iterations of Robbins-Monro algorithm */
  double  S_t, g1, g2;
  double h ;
  double sqrt_h , sqrt_rho;
  long i, ii;
  double dot1, a = 1, b = 1, payoffcarre, val_test, temp1, temp2, expo, C = 100;
  double dot2, U_1, U_2, B1_tilde, B2_tilde,  B3_tilde,  B4_tilde;
  double x_1 = 0.;
  double V_t, NormalValue1, NormalValue2, value, gamma_step;
  PnlVect *m_Mu1, *m_Mu2;

  h = T / step_number; /* discretization step */
  sqrt_h = sqrt(h);
  sqrt_rho = sqrt(1. - SQR(rho));
  m_Mu1 = pnl_vect_create_from_double(RM + 1, 0);
  m_Mu2 = pnl_vect_create_from_double(RM + 1, 0);
```

```
/* Initialisation du générateur */
pnl_rand_init(type_generator, step_number, 2 * step_number);

*theta1 = 0;
*theta2 = 0;

for (ii = 0; ii < RM; ii++)
  {
    S_t = s; /* Spot initialisation */
    V_t = sigma0; /* Variance initialisation */
    NormalValue1 = 0.0;
    NormalValue2 = 0.0;
    U_1 = 0.;
    U_2 = 0.;
    B1_tilde = pnl_rand_normal(type_generator);
    B2_tilde = pnl_rand_normal(type_generator);
    B3_tilde = pnl_rand_normal(type_generator);
    B4_tilde = pnl_rand_normal(type_generator);
    for (i = 0 ; i < step_number ; i++)
      {
        g1 = pnl_rand_normal(type_generator);
        NormalValue1 += g1;
        S_t *= (1 + (r - divid) * h + sqrt(V_t) * sqrt_h * g1); /* Price simul
        g2 = pnl_rand_normal(type_generator);
        NormalValue2 += g2;
        value = rho * g1 + sqrt_rho * g2;
        V_t = V_t + k * (theta - V_t) * h + sigma2 * sqrt_h * sqrt(V_t) * valu
        V_t = fabs(V_t); /* Absolute value of V */
      }

    U_1 = U_1 * (1 + r * h) + sqrt(V_t) * (U_1 * sqrt_h * NormalValue1 - sqrt(
    U_2 = U_2 * (1 - k * h) + ((rho * sigma2) / (2 * sqrt(V_t))) * (U_2 * sqrt

    dot2 = GET(m_Mu1, ii) * GET(m_Mu1, ii) + GET(m_Mu2, ii) * GET(m_Mu2, ii);
    dot1 = sqrt_h * NormalValue1 * GET(m_Mu1, ii) + sqrt_h * NormalValue2 * GE
    if (flag_call == 1)
      payoffcarre = SQR(MAX(S_t - Strike, 0)) + SQR(PNL_SIGN(Strike - S_t) * U
    else
      payoffcarre = SQR(MAX(Strike - S_t, 0)) + SQR(PNL_SIGN(S_t - Strike) * U
    expo = exp(-dot1 + 0.5 * dot2 * T);
```

```
      temp1 = (GET(m_Mu1, ii) - sqrt_h * NormalValue1) * expo * payoffcarre;
      temp2 = (GET(m_Mu2, ii) - sqrt_h * NormalValue2) * expo * payoffcarre;

      gamma_step = a / (b + ii);
      val_test = SQR(GET(m_Mu1, ii) - gamma_step * temp1) + SQR(GET(m_Mu2, ii) -
      val_test = sqrt(val_test);
      /* if the (n+1)th estimator of theta is into the compact C*/
      if (val_test < C)
        {
          LET(m_Mu1, ii + 1) = GET(m_Mu1, ii) - gamma_step * temp1;
          LET(m_Mu2, ii + 1) = GET(m_Mu2, ii) - gamma_step * temp2;
        }
      /* else reinitialize theta  */
      else
        {
          LET(m_Mu1, ii + 1) = x_1;
          LET(m_Mu2, ii + 1) = x_1;
          C += 1;
        }

    }

  *theta1 = GET(m_Mu1, RM);
  *theta2 = GET(m_Mu2, RM);

  pnl_vect_free(&m_Mu1);
  pnl_vect_free(&m_Mu2);

  return;
}

static int MCTwoLevel_ImportanceSampling(double s, NumFunc_1  *p, double T, doub
{
  long i, imc, j;
  double price_sample, mean_price, var_price, price_sample1, mean_price1, var_pr
  double mean_delta, delta_sample, delta_sample1, delta_sample2, mean_delta1, va
  double dot1 = 0., dot2 = 0.;
  double  S_t, S1_t, S2_t, g1, g2;
  long nb_MC, nb_MC_1;
  int m;
  double h, H;
```

```
double ptprice1, ptprice2, pterror_price1, pterror_price2;
double sqrt_h , sqrt_H, sqrt_rho;
int step_number;
double V_t, V1_t, V2_t, value, NormalValue1, NormalValue2;
double theta1, theta2;
PnlVect *B1, *B2;

double Strike;
double alpha, z_alpha;
double pterror_price;
int flag_call;

if ((p->Compute) == &Call)
  flag_call = 1;
else
  flag_call = 0;

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

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

m = (int)(sqrt(n));
step_number = m;
nb_MC = pow(m, 4.);
nb_MC_1 = CUB(m);
h = T / (SQR(m)); /* fine step of discretization */
H = T / m; /* coarse step of discretization */
sqrt_h = sqrt(h);
sqrt_H = sqrt(H);
sqrt_rho = sqrt(1. - SQR(rho));
B1 = pnl_vect_create_from_double(SQR(step_number) + 1, 0);
B2 = pnl_vect_create_from_double(SQR(step_number) + 1, 0);

/*Initialisation*/
mean_price = 0.;
var_price = 0.;
mean_delta = 0.;
var_delta = 0.;
```

```
/* Calculus of the optimal theta */
Simul_theta(flag_call, generator, Strike, 100, T, s, r, divid, V0, k, theta, s

dot2 = SQR(theta1) + SQR(theta2);
/* First Monte Carlo Sampling nb_MC=(n^2) */
for (imc = 1; imc <= nb_MC; imc++)
  {
    /* Begin of the nb_MC iterations */
    S_t = s;
    V_t = V0;
    NormalValue1 = 0.;
    NormalValue2 = 0.;
    /* Discretization using the coarse step*/
    for (i = 0 ; i < step_number ; i++)
      {
        g1 = pnl_rand_normal(generator);
        NormalValue1 += g1;
        S_t = S_t * (1 + (r - divid) * H + sqrt(V_t) * sqrt_H * (g1 + sqrt_H *
        g2 = pnl_rand_normal(generator);
        NormalValue2 += g2;
        value = rho * (g1 + sqrt_H * theta1) + sqrt_rho * (g2 + sqrt_H * theta
        V_t = V_t + k * (theta - V_t) * H + sigma2 * sqrt_H * sqrt(V_t) * valu
        V_t = fabs(V_t);
      }


    dot1 = theta1 * sqrt_H * NormalValue1 + theta2 * sqrt_H * NormalValue2;
    /*Call Price*/
    if (flag_call == 1)
      {
        price_sample = exp(-r * T) * MAX(S_t - Strike, 0) * exp(-dot1 - 0.5 *
        if (S_t - Strike > 0.0)
          delta_sample = exp(-r * T) * (S_t / s) * exp(-dot1 - 0.5 * dot2 * T)
        else  delta_sample = 0.;

      }
    else //Put Price
      {
        price_sample = exp(-r * T) * MAX(Strike - S_t, 0) * exp(-dot1 - 0.5 *
        if (Strike - S_t > 0.0)
          delta_sample = exp(-r * T) * (-S_t / s) * exp(-dot1 - 0.5 * dot2 * T
```

```
        else  delta_sample = 0.;
      }

    /* mean */
    mean_price += price_sample / nb_MC;
    mean_delta += delta_sample / nb_MC;

    /* Variance */
    var_price += SQR(price_sample - mean_price) / nb_MC;
    var_delta += SQR(delta_sample - mean_delta) / nb_MC;
  }
/* End of the nb_MC iterations */

mean_price1 = 0.;
var_price1 = 0.;
mean_delta1 = 0.;
var_delta1 = 0.;

/*Second Monte Carlo sampling (nb_MC_1=n^(3/2))*/
for (imc = 1; imc <= nb_MC_1; imc++)
  {
    /* Begin of the nb_MC_1 iterations */
    S1_t = s;
    S2_t = s;
    V1_t = V0;
    V2_t = V0;
    /* Discretization using the fine step */
    for (i = 0 ; i < SQR(step_number) ; i++)
      {
        g1 = pnl_rand_normal(generator);
        LET(B1, i + 1) = GET(B1, i) + sqrt_h * g1;
        S1_t = S1_t * (1 + (r - divid) * h + sqrt(V1_t) * ((GET(B1, i + 1) - G
        g2 = pnl_rand_normal(generator);
        LET(B2, i + 1) = GET(B2, i) + sqrt_h * g2;
        value = rho * ((GET(B1, i + 1) - GET(B1, i)) + h * theta1) + sqrt_rho
        V1_t = V1_t + k * (theta - V1_t) * h + sigma2 * sqrt(V1_t) * value;
        V1_t = fabs(V1_t);
      }

    /* Discretization using the coarse step */
    for (j = 0 ; j < step_number ; j++)
```

```
      {
        S2_t = S2_t * (1 + (r - divid) * H + sqrt(V2_t) * ((GET(B1, step_numbe
        value = rho * ((GET(B1, step_number * (j + 1)) - GET(B1, step_number *
        V2_t = V2_t + k * (theta - V2_t) * H + sigma2 * sqrt(V2_t) * value;
        V2_t = fabs(V2_t);
      }
    dot1 = theta1 * GET(B1, SQR(step_number)) + theta2 * GET(B2, SQR(step_numb
    /* Call price of the difference */
    if (flag_call == 1)
      {
        price_sample1 = exp(-r * T) * (MAX(S1_t - Strike, 0) - MAX(S2_t - Stri
        /*Delta Call 1*/
        if (S1_t - Strike > 0.0)
          delta_sample1 = exp(-r * T) * (S1_t / s) * exp(-dot1 - 0.5 * dot2 *
        else  delta_sample1 = 0.;
        /*Delta Call 2*/
        if (S2_t - Strike > 0.0)
          delta_sample2 = exp(-r * T) * (S2_t / s) * exp(-dot1 - 0.5 * dot2 *
        else  delta_sample2 = 0.;
      }
    else//Put Case
      {
        price_sample1 = exp(-r * T) * (MAX(-S1_t + Strike, 0) - MAX(-S2_t + St
        /*Delta Put 1*/
        if (Strike - S1_t > 0.0)
          delta_sample1 = exp(-r * T) * (-S1_t / s) * exp(-dot1 - 0.5 * dot2 *
        else  delta_sample1 = 0.;
        /*Delta Put 2*/
        if (Strike - S2_t > 0.0)
          delta_sample2 = exp(-r * T) * (-S2_t / s) * exp(-dot1 - 0.5 * dot2 *
        else  delta_sample2 = 0.;
      }
    /* mean */
    mean_price1 += price_sample1 / nb_MC_1;
    mean_delta1 += (delta_sample1 - delta_sample2) / nb_MC_1;

    /* variance */
    var_price1 += SQR(price_sample1 - mean_price1) / nb_MC_1;
    var_delta1 += SQR(delta_sample1 - delta_sample2 - mean_delta1) / nb_MC_1;
  }
/* End of the nb_MC_1 iterations */
```

```
  /* Statistical Romberg Price  */
  ptprice1 = mean_price;
  pterror_price1 = sqrt(var_price) / (sqrt(nb_MC) - 1);
  ptprice2 = mean_price1;
  pterror_price2 = sqrt(var_price1) / (sqrt(nb_MC) - 1);

  *ptprice = ((ptprice1) + (ptprice2)); /* European price using the Statistical
  pterror_price = pterror_price1 + pterror_price2; /* Length of 95% confidence i


  /* Price Confidence Interval */
  *inf_price = *ptprice - z_alpha * pterror_price;
  *sup_price = *ptprice + z_alpha * pterror_price;

  /* Delta estimator */
  ptdelta = mean_delta;
  pterror_delta = sqrt(var_delta) / (sqrt(nb_MC) - 1);
  ptdelta1 = mean_delta1;
  pterror_delta1 = sqrt(var_delta1) / (sqrt(nb_MC) - 1);
  *ptdelta2 = ptdelta + ptdelta1;
  pterror_delta2 = pterror_delta + pterror_delta1;

  /* Delta Confidence Interval */
  *inf_delta = *ptdelta2 - 1.96 * (pterror_delta2);
  *sup_delta = *ptdelta2 + 1.96 * (pterror_delta2);

  pnl_vect_free(&B1);
  pnl_vect_free(&B2);


  return OK;
}


int CALC(MC_TwoLevel_ImportanceSampling)(void *Opt, void *Mod, PricingMethod *Me
{
  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 MCTwoLevel_ImportanceSampling(ptMod->S0.Val.V_PDOUBLE,
                                       ptOpt->PayOff.Val.V_NUMFUNC_1,
                                       ptOpt->Maturity.Val.V_DATE - ptMod->T.Val
                                       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_INT,
                                       Met->Par[1].Val.V_ENUM.value,
                                       Met->Par[2].Val.V_PDOUBLE,
                                       &(Met->Res[0].Val.V_DOUBLE),
                                       &(Met->Res[1].Val.V_DOUBLE),
                                       &(Met->Res[2].Val.V_DOUBLE),
                                       &(Met->Res[3].Val.V_DOUBLE),
                                       &(Met->Res[4].Val.V_DOUBLE),
                                       &(Met->Res[5].Val.V_DOUBLE)
                                      );

}

static int CHK_OPT(MC_TwoLevel_ImportanceSampling)(void *Opt, void *Mod)
{

  if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt
    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_INT = 400;
      Met->Par[1].Val.V_ENUM.value = 0;
      Met->Par[1].Val.V_ENUM.members = &PremiaEnumRNGs;
      Met->Par[2].Val.V_DOUBLE = 0.95;
    }
  return OK;
}


PricingMethod MET(MC_TwoLevel_ImportanceSampling) =
{
  "MC_TwoLevel_ImportanceSampling",
  {
    {"TimeStepNumber", LONG, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Confidence Value", DOUBLE, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_TwoLevel_ImportanceSampling),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Delta", DOUBLE, {100}, FORBID} ,
    {"Inf Price", DOUBLE, {100}, FORBID},
    {"Sup Price", DOUBLE, {100}, FORBID} ,
    {"Inf Delta", DOUBLE, {100}, FORBID},
    {"Sup Delta", DOUBLE, {100}, FORBID} ,
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_TwoLevel_ImportanceSampling),
  CHK_mc,
  MET(Init)
};
```