Help

```
#include "
href../../../mod/hawkes_trading/hawkes_trading_stdt/hawkes_trading_stdt_h_src.pd
#include "
href../../../opt/stdt/stdt_h_src.pdfstdt/stdt.h"

#define N_MAX_JUMPS 100000
#define N_MAX_ITER_SIMU 100000000
#define N_MAX_STEPS_STRAT 1000000
#define NUM_ZERO 0.00000001

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2015 + 2) //The "#e
static int CHK_OPT(MC_AlfonsiBlanc)(void *Opt, void *Mod)
{
  return NONACTIVE;
}

int CALC(MC_AlfonsiBlanc)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

//MC
static double zeta(double y)
{
  if (fabs(y) < NUM_ZERO)
    return 1.0;
  else
    return (1.0 - exp(-y)) / y;
}

//MX
static double omega(double y)
{
  if (fabs(y) < NUM_ZERO)
    return 0.5;
  else
    return (exp(-y) - 1.0 + y) / (y * y);
}
```

```
/*////////////////////////////////////////////////*/
//MC
static double LawMu(double m1)
{
  double u;
  u = -(rand() + 1) / (double)(RAND_MAX + 1);
  return -log(u) * m1;
}

/*////////////////////////////////////////////////*/
// Simulation of a trajectory of the Hawks process
static void Simu_Price(double *jump_times, double *volumes, double T, double k_i
{

  long j;
  long cur_ind_jump;
  double kmax_jump;
  double current_kp;
  double current_km;
  int is_buy;
  int is_sell;
  double t_proc;
  double order_interval;
  double U;
  double dN;

  cur_ind_jump = 0;
  j = 0;

  jump_times[0] = 0;
  volumes[0] = 0;
  kmax_jump = MAX(kp0, k_infty) + MAX(km0, k_infty);
  current_kp = kp0;
  current_km = km0;
  is_buy = 0;
  is_sell = 0;
  t_proc = 0;

  while ((t_proc < T) && (j < N_MAX_ITER_SIMU) && (cur_ind_jump < N_MAX_JUMPS))
    {
```

```
      order_interval = pnl_rand_uni(generator);
      order_interval = -log(order_interval) / kmax_jump;
      t_proc = t_proc + order_interval;

      current_kp = k_infty + (current_kp - k_infty) * exp(-beta * order_interval
      current_km = k_infty + (current_km - k_infty) * exp(-beta * order_interval
      U = pnl_rand_uni(generator);
      is_buy = (U <= (current_kp / kmax_jump)) ? 1 : 0;
      is_sell = ((U > (current_kp / kmax_jump)) && (U <= ((current_kp + current_

      if ((t_proc < T) && (is_buy || is_sell))
        {
          dN = pnl_rand_exp(1. / m1, generator);
          if (is_buy)
            {
              current_kp += ios;
              current_km += ioc;
            }
          if (is_sell)
            {
              dN = -1.0 * dN;
              current_kp += ioc;
              current_km += ios;
            }
          cur_ind_jump++;
          jump_times[cur_ind_jump] = t_proc;
          volumes[cur_ind_jump] = dN;
        }

      kmax_jump = MAX(current_kp + current_km, 2 * k_infty);
      j++;
    }

  cur_ind_jump++;
  jump_times[cur_ind_jump] = T;
  volumes[cur_ind_jump] = 0;

  jump_times[cur_ind_jump + 1] = -1.0;

  return;
}
```

```
/*////////////////////////////////////////////////*/
//Compute the optimal strategy and its cost
static double Opt_Strat(double *trade_times, double *strat, double *prec_price,
{

  double *jump_times;
  double *volumes;
  double cost;
  double Dt, St, delta_t;
  double prec_t, cur_t;
  double h;
  double Xt, dXt;
  long cur_ind_strat, cur_ind_jump;
  double eta;
  double Tmt;
  double quad_cost;

  cost = 0.0;
  prec_t = 0.0;
  cur_t = 0.0;
  Dt = D0;
  St = S0;
  delta_t = kp0 - km0;
  h = MAX(T / (N_MAX_STEPS_STRAT + 1), discr_step);
  Xt = X0;
  dXt = 0.0;
  cur_ind_strat = 0;
  cur_ind_jump = 0;
  eta = beta - ios + ioc;
  Tmt = T;
  quad_cost = 0.0;

  jump_times = (double *)calloc(N_MAX_JUMPS, sizeof(double));
  volumes = (double *)calloc(N_MAX_JUMPS, sizeof(double));

  Simu_Price(jump_times, volumes, T, k_infty, beta, ios, ioc, kp0, km0, m1, gene

  quad_cost = 1.0;
  prec_price[cur_ind_strat] = Dt + St;
  trade_times[cur_ind_strat] = cur_t;
```

```
Tmt = T - cur_t;
dXt = -(1.0 - epsilon) * Xt;
dXt += delta_t * m1 * (2.0 + rho * Tmt * (1.0 + zeta(eta * Tmt) + nu * rho * T
dXt -= (1.0 + rho * Tmt) * q * Dt;
dXt /= (1.0 - epsilon) * (2.0 + rho * Tmt);
Xt += dXt;
cost += (Dt + St) * dXt + 0.5 * quad_cost * dXt * dXt / q;
strat[cur_ind_strat] = dXt;
Dt += dXt * (1.0 - epsilon) / q;
St += dXt * epsilon / q;

while (cur_t < (T - h))
  {
    prec_t = cur_t;
    cur_ind_strat++;
    if (jump_times[cur_ind_jump + 1] < (cur_t + h))
      {
        cur_ind_jump++;
        cur_t = jump_times[cur_ind_jump];
        Dt = Dt * exp(-rho * (cur_t - prec_t)) + volumes[cur_ind_jump] * (1.0
        St += volumes[cur_ind_jump] * nu / q;
        delta_t *= exp(-beta * (cur_t - prec_t));
        if (volumes[cur_ind_jump] > 0) delta_t += ios - ioc;
        if (volumes[cur_ind_jump] < 0) delta_t -= ios - ioc;
        quad_cost = 1.0;
      }
    else
      {
        cur_t += h;
        Dt *= exp(-rho * (cur_t - prec_t));
        delta_t *= exp(-beta * (cur_t - prec_t));
        quad_cost = 0.0;
      }
    prec_price[cur_ind_strat] = Dt + St;
    trade_times[cur_ind_strat] = cur_t;
    Tmt = T - cur_t;
    dXt = -(1.0 - epsilon) * Xt;
    dXt += delta_t * m1 * (2.0 + rho * Tmt * (1 + zeta(eta * Tmt) + nu * rho *
    dXt -= (1.0 + rho * Tmt) * q * Dt;
    dXt /= (1.0 - epsilon) * (2.0 + rho * Tmt);
    Xt += dXt;
```

```
      cost += (Dt + St) * dXt + 0.5 * quad_cost * dXt * dXt / q;
      strat[cur_ind_strat] = dXt;
      Dt += dXt * (1.0 - epsilon) / q;
      St += dXt * epsilon / q;
    }

  quad_cost = 1.0;
  prec_t = cur_t;
  cur_t = T;
  cur_ind_strat++;
  Dt *= exp(-rho * (cur_t - prec_t));
  delta_t *= exp(-beta * (cur_t - prec_t));
  prec_price[cur_ind_strat] = Dt + St;
  trade_times[cur_ind_strat] = cur_t;
  dXt = -Xt;
  Xt = 0.0;
  cost += (Dt + St) * dXt + 0.5 * quad_cost * dXt * dXt / q;
  strat[cur_ind_strat] = dXt;
  Dt += dXt * (1.0 - epsilon) / q;
  St += dXt * epsilon / q;

  free(jump_times);
  free(volumes);

  return cost;
}

/*////////////////////////////////////////////////*/
static double MC_Cost(double T, double X0, double rho, double nu, double epsilon
{

  double *trade_times;
  double *strat;
  double *prec_price;
  double cost;
  double mean_cost, sum_cost, sum_cost2;
  long i;
  double alpha, z_alpha;
  double cost_sd;

  pnl_rand_sseed(generator, 0);
```

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

  sum_cost = 0.0;
  sum_cost2 = 0.0;

  for (i = 0; i < npaths; i++)
    {
      trade_times = (double *)calloc(N_MAX_STEPS_STRAT, sizeof(double));
      strat = (double *)calloc(N_MAX_STEPS_STRAT, sizeof(double));
      prec_price = (double *)calloc(N_MAX_STEPS_STRAT, sizeof(double));
      cost = Opt_Strat(trade_times, strat, prec_price, discr_step, T, X0, rho, n
      sum_cost += cost;
      sum_cost2 += cost * cost;
      free(trade_times);
      free(strat);
      free(prec_price);
    }
  mean_cost = sum_cost / npaths;

  cost_sd = sqrt((sum_cost2 / npaths - mean_cost * mean_cost) * npaths / (npaths

  *ptprice = mean_cost;
  *inf_price = *ptprice - z_alpha * cost_sd / sqrt(npaths);
  *sup_price = *ptprice + z_alpha * cost_sd / sqrt(npaths);

  return OK;
}

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

  return MC_Cost(ptOpt->Maturity.Val.V_DATE, ptOpt->InitialPosition.Val.V_PDOUBL
                 Met->Par[1].Val.V_SPDOUBLE, Met->Par[2].Val.V_ENUM.value, Met->
}

static int CHK_OPT(MC_AlfonsiBlanc)(void *Opt, void *Mod)
{
```

```
  if ((strcmp(((Option *)Opt)->Name, "OptimalExecution") == 0))
    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 = 1000;
      Met->Par[1].Val.V_SPDOUBLE = 0.0002778;
      Met->Par[2].Val.V_ENUM.value = 0;
      Met->Par[2].Val.V_ENUM.members = &PremiaEnumMCRNGs;
      Met->Par[3].Val.V_SPDOUBLE = 0.95;
    }

  return OK;
}

PricingMethod MET(MC_AlfonsiBlanc) = {
  "MC_AlfonsiBlanc",
  { { "N iterations", LONG, { 100 }, ALLOW },
    { "Discretization step", SPDOUBLE, { 100 }, ALLOW },
    { "RandomGenerator", ENUM, { 100 }, ALLOW },
    { "Confidence Value", DOUBLE, { 100 }, ALLOW },
    { " ", PREMIA_NULLTYPE, { 0 }, FORBID } },
  CALC(MC_AlfonsiBlanc),
  { { "Cost", DOUBLE, { 100 }, FORBID },
    { "Inf Cost", DOUBLE, { 100 }, FORBID },
    { "Sup Cost", DOUBLE, { 100 }, FORBID },
    { " ", PREMIA_NULLTYPE, { 0 }, FORBID } },
  CHK_OPT(MC_AlfonsiBlanc),
  CHK_ok,
  MET(Init)
};
```