Help

```
#include <stdlib.h>
#include  "kou1d_lim.h"
#include "pnl/pnl_random.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2015+2) //The "#els
static int CHK_OPT(MC_mcwhout_kou)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_mcwhout_kou)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static int mcwh_kou_bar(int b_type, int ifCall, double S0, double sigma, double
                        double r, double divid,
                        double T,  double K,
                        double bar, double rebate,
                        int generator, long int n_paths, int n_points,
                        double *ptprice, double *priceError)
{

  double log_l_S0, payoff, sum_payoff, sum_square_payoff, var_payoff;
  double discount;
  long int i;
  int j;

  double nu = ((r - divid) - sigma * sigma / 2 - lambda * (p * lambdap / (lambda
  double q = lambda + n_points / T; //lambda+gamma in Theorem 4, [Kuznetsov et a
  double Pb = (q - lambda) / q; // Bernoulli probability
  double xin, bn, Vn, V0, Jn, J0, Sn, In; // the functions in Theorem 4, [Kuznet
  double Dis, betam, betap;
  double dt, tn; // time variables
  PnlRng *rng = PnlRngArray[generator];
  pnl_rng_sseed(rng, 0);


  Vn = 0;
```

```
Jn = 0;
sigma = sigma * sigma;
Dis = sqrt(nu * nu + 2 * sigma * q);
betam = (Dis - nu) / sigma; // exponential coefficient for the plus factor
betap = (nu + Dis) / sigma; // exponential coefficient for the minus factor
dt = T / n_points;
discount = exp(-r * T);
log_l_S0 = log(bar / S0);

sum_payoff = 0;
sum_square_payoff = 0;


/*Down Case*/
if (b_type == 0)
  {
    if (ifCall == 1) //call
      {
        for (i = 0; i < n_paths; i++)
          {
            Vn = 0;
            Jn = 0;
            tn = 0;
            for (j = 1; j <= n_points; j++)
              {
                V0 = Vn;
                J0 = Jn;
                xin = pnl_rng_dblexp(lambdap, lambdam, p, rng);
                bn = pnl_rng_bernoulli(Pb, rng);
                Sn = pnl_rng_exp(betam, rng);
                In = -pnl_rng_exp(betap, rng);
                Vn = Vn + Sn + In + xin * (1 - bn);
                Jn = MIN(J0, MIN(Vn, V0 + In));
                tn = tn + dt;
                if (Jn <= log_l_S0)
                  {
                    j = n_points + 2;
                  }
              }
            payoff = discount * (S0 * exp(Vn) - K) * (S0 * exp(Vn) > K) * (Jn
            sum_payoff += payoff;
```

```
            sum_square_payoff += payoff * payoff;
          }
        var_payoff = (sum_square_payoff - sum_payoff * sum_payoff / n_paths) /
        *ptprice = sum_payoff / n_paths;
        *priceError = 1.96 * sqrt(var_payoff) / sqrt((double)n_paths);
      }
    if (ifCall == 0) //put
      {
        for (i = 0; i < n_paths; i++)
          {
            Vn = 0;
            Jn = 0;
            tn = 0;
            for (j = 1; j <= n_points; j++)
              {
                V0 = Vn;
                J0 = Jn;
                xin = pnl_rng_dblexp(lambdap, lambdam, p, rng);
                bn = pnl_rng_bernoulli(Pb, rng);
                Sn = pnl_rng_exp(betam, rng);
                In = -pnl_rng_exp(betap, rng);
                Vn = Vn + Sn + In + xin * (1 - bn);
                Jn = MIN(J0, MIN(Vn, V0 + In));
                tn = tn + dt;
                if (Jn <= log_l_S0)
                  {
                    j = n_points + 2;
                  }
              }

            payoff = discount * (K - S0 * exp(Vn)) * (S0 * exp(Vn) < K) * (Jn
            sum_payoff += payoff;
            sum_square_payoff += payoff * payoff;
          }
        var_payoff = (sum_square_payoff - sum_payoff * sum_payoff / n_paths) /
        *ptprice = sum_payoff / n_paths;
        *priceError = 1.96 * sqrt(var_payoff) / sqrt((double)n_paths);
      }
  }
/*Up Case*/
if (b_type == 1)
```

```
{
  if (ifCall == 1) //call
    {
      for (i = 0; i < n_paths; i++)
        {
          Vn = 0;
          Jn = 0;
          tn = 0;
          for (j = 1; j <= n_points; j++)
            {
              V0 = Vn;
              J0 = Jn;
              xin = pnl_rng_dblexp(lambdap, lambdam, p, rng);
              bn = pnl_rng_bernoulli(Pb, rng);
              Sn = pnl_rng_exp(betam, rng);
              In = -pnl_rng_exp(betap, rng);
              Vn = Vn + Sn + In + xin * (1 - bn);
              Jn = MAX(J0, MAX(Vn, V0 + Sn));
              tn = tn + dt;
              if (Jn >= log_l_S0)
                {
                  j = n_points + 2;
                }
            }

          payoff = discount * (S0 * exp(Vn) - K) * (S0 * exp(Vn) > K) * (Jn
          sum_payoff += payoff;
          sum_square_payoff += payoff * payoff;
        }
      var_payoff = (sum_square_payoff - sum_payoff * sum_payoff / n_paths) /
      *ptprice = sum_payoff / n_paths;
      *priceError = 1.96 * sqrt(var_payoff) / sqrt((double)n_paths);
    }
  if (ifCall == 0) //put
    {
      for (i = 0; i < n_paths; i++)
        {
          Vn = 0;
          Jn = 0;
          tn = 0;
          for (j = 1; j <= n_points; j++)
```

```
                {
                  V0 = Vn;
                  J0 = Jn;
                  xin = pnl_rng_dblexp(lambdap, lambdam, p, rng);
                  bn = pnl_rng_bernoulli(Pb, rng);
                  Sn = pnl_rng_exp(betam, rng);
                  In = -pnl_rng_exp(betap, rng);
                  Vn = Vn + Sn + In + xin * (1 - bn);
                  Jn = MAX(J0, MAX(Vn, V0 + Sn));
                  tn = tn + dt;
                  if (Jn >= log_l_S0)
                    {
                      j = n_points + 2;
                    }
                }

              payoff = discount * (K - S0 * exp(Vn)) * (S0 * exp(Vn) < K) * (Jn
              sum_payoff += payoff;
              sum_square_payoff += payoff * payoff;
            }
          var_payoff = (sum_square_payoff - sum_payoff * sum_payoff / n_paths) /
          *ptprice = sum_payoff / n_paths;
          *priceError = 1.96 * sqrt(var_payoff) / sqrt((double)n_paths);
        }
    }

  return OK;
}

//=============================================================================
int CALC(MC_mcwhout_kou)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid, limit, strike, spot, rebate;

  NumFunc_1 *p;
  int res;
  int upor;
  int ifCall;
```

```
  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
  limit = ((ptOpt->Limit.Val.V_NUMFUNC_1)->Compute)((ptOpt->Limit.Val.V_NUMFUNC_
  p = ptOpt->PayOff.Val.V_NUMFUNC_1;
  strike = p->Par[0].Val.V_DOUBLE;
  spot = ptMod->S0.Val.V_DOUBLE;
  ifCall = ((p->Compute) == &Call);

  rebate = ((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compute)((ptOpt->Rebate.Val.V_NUMFU


  if ((ptOpt->DownOrUp).Val.V_BOOL == DOWN)
    upor = 0;
  else upor = 1;

  res = mcwh_kou_bar(upor, ifCall, spot, ptMod->Sigma.Val.V_PDOUBLE, ptMod->Lamb
                     r, divid,
                     ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, strike,
                     limit, rebate,
                     Met->Par[0].Val.V_ENUM.value, Met->Par[1].Val.V_INT2, Met->
                     &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE));

  return res;

}
static int CHK_OPT(MC_mcwhout_kou)(void *Opt, void *Mod)
{
  Option *ptOpt = (Option *)Opt;
  TYPEOPT *opt = (TYPEOPT *)(ptOpt->TypeOpt);

  if ((opt->OutOrIn).Val.V_BOOL == OUT)
    if ((opt->Parisian).Val.V_BOOL == FALSE)
      if ((opt->EuOrAm).Val.V_BOOL == EURO)
        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_ENUM.value = 0;
      Met->Par[0].Val.V_ENUM.members = &PremiaEnumMCRNGs;
      Met->Par[1].Val.V_LONG = 100000;
      Met->Par[2].Val.V_PINT = 200;

    }
  return OK;
}

PricingMethod MET(MC_mcwhout_kou) =
{
  "MC_WHBar_Kou",
  { {"RandomGenerator", ENUM, {100}, ALLOW},
    {"N iterations", LONG, {100}, ALLOW},
    {"Number of discretization steps", LONG, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_mcwhout_kou),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Price_Error", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_mcwhout_kou),
  CHK_split,
  MET(Init)
};
```