Help

```
#include <stdlib.h>
#include <math.h>
#include  "temperedstable1d_lim.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_fft.h"
#include "math/wienerhopf.h"

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

static int froot(long int kmax, PnlVect *F, double Fx, long int *k0, long int *k
{
  long int k;
  *k0 = 0;
  *k1 = kmax - 1;
  if (Fx < GET(F, 0))
    {
      *k1 = *k0;
    }
  while (*k1 > *k0 + 1)
    {
      k = ((*k0 + *k1) / 2);
      if (Fx > GET(F, k))
        {
          *k0 = k;
        }
      else
        {
          *k1 = k;
        }
    }
```

```
  return 1;
}
//=====================================================================================
static int mc_wh_tsl_downout(int am, int b_type, int ifCall, double S0, double l
                             double num, double nup, double cm, double cp,
                             double r, double divid,
                             double T, double h, double K,
                             double bar, double rebate,
                             double er, long int n_points, long int n_paths, int
                             double *ptPrice, double *priceError)
{
  double cnup, cnum, lpnu, lmnu, mu, q, kx;
  double log_l_S0, payoff, sum_payoff, sum_square_payoff, var_payoff;
  double discount;
  long int i, kmax, Nmax, k0, k1;
  int j;
  double Vn, V0, Jn, J0, Sn, In; // the functions in Theorem 1, [Kuznetsov et al
  double dt, tn; // time variables
  PnlVect  *y, *Fp, *Fm, *tp1, *tm1, *alin1, *al1;
  PnlRng *rng;

  cnup = cp * pnl_tgamma(-nup);
  cnum = cm * pnl_tgamma(-num);

  lpnu = exp(nup * log(lp1));
  lmnu = exp(num * log(-lm1));

  mu = r - divid + cnup * (lpnu - exp(nup * log(lp1 + 1.0))) + cnum * (lmnu - ex

  q = n_points / T; //lambda in Theorem 1, [Kuznetsov et al, 2011]; the shift qu
  Vn = 0;
  Jn = 0;
  dt = T / n_points;
  discount = exp(-r * T);
  log_l_S0 = log(bar / S0);

  sum_payoff = 0;
  sum_square_payoff = 0;

  rng = PnlRngArray[gen];
```

```
pnl_rng_sseed(rng, 0);

j = 64;
kx = er * log(2.0) / h;
while (j < kx)
  {
    j = j * 2;
  }
kmax = j;
Nmax = 2 * kmax; //  the number of the space points for the cdfs

//Memory allocation for space grid
y = pnl_vect_create_from_zero(Nmax + 1); //space grid points
Fp = pnl_vect_create_from_zero(kmax + 1); // CDF for the Plus WH-factor
Fm = pnl_vect_create_from_zero(kmax + 1); // CDF for the Plus WH-factor
al1 = pnl_vect_create_from_zero(2 * kmax + 2);
alin1 = pnl_vect_create_from_zero(2 * kmax + 2);
tp1 = pnl_vect_create_from_zero(2 * kmax + 2);
tm1 = pnl_vect_create_from_zero(2 * kmax + 2);

for (j = 0; j < kmax; j++)
  {
    y->array[j] = j * h; //space grid
  }
for (j = kmax; j < Nmax; j++)
  {
    y->array[j] = (j - Nmax) * h; //space grid
  }
findcoefnew(1, mu, 0.0, lm1, lp1, num, nup, cnum, cnup, q, r, T, h, kmax, er,

for (j = 0; j < Nmax; j++) alin1->array[j] = al1->array[j + 1];
findfactor(0.0, 0.0, h,  kmax, lp1, lm1, alin1, tp1, tm1);// WH factorization:
//////////////////cdfs calculations////////////////////////////////
fft_real(tp1, 2 * kmax, 1);
fft_real(tm1, 2 * kmax, 1);
LET(Fp, 0) = GET(tp1, 0);
for (j = 1; j < kmax; j++)
  {
    LET(Fp, j) = GET(Fp, j - 1) + GET(tp1, j);
  }
LET(Fm, 0) = GET(tm1, kmax + 1);
```

```
for (j = 1; j < kmax - 1; j++)
  {
    LET(Fm, j) = GET(Fm, j - 1) + GET(tm1, j + kmax + 1);
  }
LET(Fm, kmax - 1) = GET(Fm, kmax - 2) + GET(tm1, 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;
                Sn = pnl_rng_uni(rng); //printf("Sn = %f \ n", Sn);
                froot(kmax, Fp, Sn, &k0, &k1);
                if (k1 == 0)
                  {
                    Sn = 0.0;
                  }
                else
                  {
                    Sn = GET(y, k0) + h * (Sn - GET(Fp, k0)) / (GET(Fp, k1) -
                  }
                In = pnl_rng_uni(rng);
                froot(kmax, Fm, In, &k0, &k1);
                In = GET(y, k0 + kmax + 1) + h * (In - GET(Fm, k0)) / (GET(Fm,
                Vn = Vn + Sn + In;
                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;
                Sn = pnl_rng_uni(rng);
                froot(kmax, Fp, Sn, &k0, &k1);
                if (k1 == 0)
                  {
                    Sn = 0.0;
                  }
                else
                  {
                    Sn = GET(y, k0) + h * (Sn - GET(Fp, k0)) / (GET(Fp, k1) -
                  }
                In = pnl_rng_uni(rng);
                froot(kmax, Fm, In, &k0, &k1);
                In = GET(y, k0 + kmax + 1) + h * (In - GET(Fm, k0)) / (GET(Fm,
                Vn = Vn + Sn + In;
                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;
                Sn = pnl_rng_uni(rng);
                froot(kmax, Fp, Sn, &k0, &k1);
                if (k1 == 0)
                  {
                    Sn = 0.0;
                  }
                else
                  {
                    Sn = GET(y, k0) + h * (Sn - GET(Fp, k0)) / (GET(Fp, k1) -
                  }
                In = pnl_rng_uni(rng);
                froot(kmax, Fm, In, &k0, &k1);
                In = GET(y, k0 + kmax + 1) + h * (In - GET(Fm, k0)) / (GET(Fm,
                Vn = Vn + Sn + In;
                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;
                Sn = pnl_rng_uni(rng);
                froot(kmax, Fp, Sn, &k0, &k1);
                if (k1 == 0)
                  {
                    Sn = 0.0;
                  }
                else
                  {
                    Sn = GET(y, k0) + h * (Sn - GET(Fp, k0)) / (GET(Fp, k1) -
                  }
                In = pnl_rng_uni(rng);
                froot(kmax, Fm, In, &k0, &k1);
                In = GET(y, k0 + kmax + 1) + h * (In - GET(Fm, k0)) / (GET(Fm,
                Vn = Vn + Sn + In;
                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_whout_ts)(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 upordown;
  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)
```

```
    upordown = 0;
  else upordown = 1;

  res = mc_wh_tsl_downout(ptOpt->EuOrAm.Val.V_BOOL, upordown, ifCall, spot, -ptM
                          ptMod->AlphaPlus.Val.V_PDOUBLE, ptMod->AlphaMinus.Val.
                          ptMod->CPlus.Val.V_PDOUBLE, ptMod->CMinus.Val.V_PDOUBL
                          r, divid,
                          ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, Met-
                          limit, rebate,
                          Met->Par[0].Val.V_DOUBLE, Met->Par[2].Val.V_INT2, Met-
                          &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBL

  return res;

}

static int CHK_OPT(MC_whout_ts)(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)
{
  static int first = 1;

  if (first)
    {
      Met->HelpFilenameHint = "mc_whbar_ts";
      Met->Par[0].Val.V_PDOUBLE = 2.0;
      Met->Par[1].Val.V_PDOUBLE = 0.00001;
      Met->Par[2].Val.V_INT2 = 200;
      Met->Par[3].Val.V_LONG = 100000;
```

```
      Met->Par[4].Val.V_ENUM.value = 0;
      Met->Par[4].Val.V_ENUM.members = &PremiaEnumRNGs;

      first = 0;
    }
  return OK;
}

PricingMethod MET(MC_whout_ts) =
{
  "MC_WIENERHOPH",
  { {"Scale of logprice range", DOUBLE, {100}, ALLOW},
    {"Space Discretization Step", DOUBLE, {500}, ALLOW},
    {"TimeStepNumber", INT2, {100}, ALLOW},
    {"Number of paths", INT2, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_whout_ts),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Error", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_whout_ts),
  CHK_split,
  MET(Init)
};
```