Help

```
#include "nig1d_std.h"
#include "pnl/pnl_integration.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_mathtools.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(TR_MSS_NIG)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_MSS_NIG)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else
static double sigma_g, theta_g, kappa_g, A, B, C, dt;

//---------------------------------------------------------------
//---------------------------------------------------------------
//-Density Function NIG
//---------------------------------------------------------------
static double probdensityx(double x, void *p) // Bonne
{
  double y, bes, Cp;
  double t;
  t = dt;
  bes = pnl_bessel_k(1, B * sqrt(SQR(x) + SQR(t * sigma_g) / kappa_g));
  Cp = t * exp(t / kappa_g) * sqrt(SQR(theta_g) / kappa_g / SQR(sigma_g) + 1 / S
  y = Cp * exp(A * x) * bes / sqrt(SQR(x) + SQR(t * sigma_g) / kappa_g);

  return y;
}

static double pt(double x, double z)
{
  double abserr, results;
  int neval;
```

```
  PnlFunc func;
  func.F = probdensityx;
  func.params = NULL;
  neval = 50;
  pnl_integration_GK(&func, x, z, 0.0001, 1, &results, &abserr, &neval);

  return results;

}

static double Ldensity(double t, void *p)
{
  double y, besss;

  besss = pnl_bessel_k(1, B * fabs(t));

  y = C * exp(A * t) * besss / fabs(t);

  return y;
}

static double Levy(double x, double z)
{
  double abserr, results;
  int neval;
  PnlFunc func;
  func.F = Ldensity;
  func.params = NULL;
  neval = 500;
  pnl_integration_GK(&func, x, z, 0.0001, 1, &results, &abserr, &neval);

  return results;

}

static double omegadensity(double t, void *p)
{
  double y, b;

  b = pnl_bessel_k(1, B * fabs(t));
```

```
  if (fabs(t) <= 1)
    y = (exp(t) - 1 - t) * C * exp(A * t) * b / fabs(t);
  else
    y = (exp(t) - 1) * C * exp(A * t) * b / fabs(t);

  return y;
}


static double iomega(double x, double z)
{
  double abserr, results;
  int neval;
  PnlFunc func;
  func.F = omegadensity;
  func.params = NULL;
  neval = 500;
  pnl_integration_GK(&func, x, z, 0.0001, 1, &results, &abserr, &neval);

  return results;

}

static double Ldensityx2(double t, void *p)
{
  double y, be;

  be = pnl_bessel_k(1, B * fabs(t));

  y = C * fabs(t) * exp(A * t) * be;

  return y;
}

static double sigmabar2(double x, double z)
{
  double abserr, results;
  int neval;
  PnlFunc func;
  func.F = Ldensityx2;
  func.params = NULL;
```

```
  neval = 5000;
  pnl_integration_GK(&func, x, z, 0.0001, 1, &results, &abserr, &neval);

  return results;

}

static int TreeNIG(int am, double S0, NumFunc_1  *p, double T, double r, double
{
  double *P, *stock, *proba, *x;
  double dx;
  int i, j, k, N2, N_plus, N_minus, M;
  double exp_drift, dis, emp_mean, sum, sig, omega;

  sigma_g = sigma;
  theta_g = theta;
  kappa_g = kappa;

  //Lévy measure
  A = theta / SQR(sigma);
  B = sqrt(SQR(theta) + SQR(sigma) / kappa) / SQR(sigma);
  C = sqrt(SQR(theta) + SQR(sigma) / kappa) / (M_PI * sigma * sqrt(kappa));

  N_plus = N;
  N_minus = N;
  M = N_plus + N_minus;
  N2 = N * M;

  //Memory allocation
  P = (double *)malloc((N2 + 1) * sizeof(double));
  stock = (double *)malloc((N2 + 1) * sizeof(double));
  proba = (double *)malloc((M + 1) * sizeof(double));
  x = (double *)malloc((M + 1) * sizeof(double));

  //Time step
  dt = T / (double)N;

  //Space step
  sig = sqrt(sigmabar2(-0.1, -0.0000001) + sigmabar2(0.0000001, 0.1));
  if (flag_scheme == 1)
    dx = sig * sqrt(dt);
```

```
else
  dx = (0.5 / T) * sigma * sqrt(dt);

for (i = 0; i <= M; i++)
  proba[i] = 0.;

if (flag_scheme == 1) //Compute true transition probaiblities
  {
    sum = 0.;
    for (i = 0; i <= M; i++)
      {
        x[i] = -(double)N_minus * dx + (double)i * dx;
        if (i != M / 2)
          proba[i] = pt(x[i] - dx / 2., x[i] + dx / 2.);
        sum += proba[i];
      }
    proba[M / 2] = 1. - sum;
  }
else //Paper MLS
  {

    sum = 0.;
    for (i = 0; i <= M; i++)
      {
        x[i] = -(double)N_minus * dx + (double)i * dx;

        if (i != M / 2)
          {
            proba[i] = Levy(x[i] - dx / 2., x[i] + dx / 2.) * dt;
            sum += proba[i];
          }
      }
    proba[M / 2] = 1. - sum;
  }

//Compute expectation
emp_mean = 0.;
for (i = 0; i <= M; i++)
  if (fabs(proba[i]) <= 1)
    emp_mean += proba[i] * x[i];
```

```
//Discounted probabilities
for (i = 0; i <= M; i++)
  proba[i] *= exp(-r * dt);

/*Maturity condition*/
//Drift changement for the risk-neutral measure = -iomega(-100,100)
omega = iomega(-1, -0.0001) + iomega(0.00001, 1) + iomega(1, 20) + iomega(-20,
dis = exp(-(r - omega) * dt + emp_mean);
exp_drift = exp((r - omega) * T - (double)N * emp_mean);

for (i = 0; i <= N2; i++)
  {
    stock[i] = S0 * exp_drift * exp(-(double)N * N_minus * dx + (double)i * dx
    P[i] = (p->Compute)(p->Par, stock[i]);
  }

/*******************/
/*Backward Resolution*/
/*******************/
for (i = 1; i <= N; i++)
  {
    for (j = 0; j <= N2 - M * i; j++)
      {
        //Compute Conditional Expectation
        sum = 0.;
        for (k = 0; k <= M; k++)
          sum += proba[k] * P[j + k];
        P[j] = sum;

        //American case
        if (am)
          {
            P[j] = MAX(P[j], (p->Compute)(p->Par, stock[j + M / 2 * i] * pow(d
          }
      }

    //Delta
    if (i == N - 1)
      *ptdelta = (P[M / 2 + 1] - P[M / 2 - 1]) / (2 * S0 * dx);
  }
```

```
  //Price
  *ptprice = P[0];

  //Memory deallocation
  free(P);
  free(stock);
  free(proba);
  free(x);

  return OK;


  return OK;
}

int CALC(TR_MSS_NIG)(void *Opt, void *Mod, PricingMethod *Met)
{
  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 TreeNIG(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
                 ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Maturity.Val.V_DATE - ptM
}

static int CHK_OPT(TR_MSS_NIG)(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)
{
  static int first = 1;
```

```
  if (first)
    {
      Met->Par[0].Val.V_INT2 = 100;
      Met->Par[1].Val.V_ENUM.value = 1;
      Met->Par[1].Val.V_ENUM.members = &PremiaEnumSchemeTreeMSS;
      first = 0;
    }

  return OK;
}

PricingMethod MET(TR_MSS_NIG) =
{
  "TR_MSS_NIG",
  { {"TimeStepNumber", INT2, {100}, ALLOW},
    {"Type of tree", ENUM, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(TR_MSS_NIG),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID}, {" ", PRE
  CHK_OPT(TR_MSS_NIG),
  CHK_split,
  MET(Init)
};
```