Help

```
#include "
href../../../mod/variancegamma1d/variancegamma1d_std/variancegamma1d_std_h_src.p
#include "pnl/pnl_integration.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_mathtools.h"
#include "
href../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(TR_MSS_VG)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_MSS_VG)(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 VG
//---------------------------------------------------------------
static double probdensityx(double x, void *p)
{
  double val, esp, y, bes;
  double t;

  t = dt;
  val = fabs(SQR(x) / (SQR(theta_g) + 2.*SQR(sigma_g) / kappa_g));
  esp = t / kappa_g / 2. - 0.25;
  bes = pnl_bessel_k(t / kappa_g - 0.5, B * fabs(x));

  y = (C * pow(val, esp) * exp(A * x) * bes) / (pnl_tgamma(t / kappa_g) * pow(ka

  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;

  y = exp(A * t - B * fabs(t)) / (kappa_g * 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;
  if (fabs(t) <= 1)
    y = (exp(t) - 1 - t) * exp(A * t - B * fabs(t)) / (kappa_g * fabs(t));
  else
    y = (exp((A + 1) * t - B * fabs(t)) - exp(A * t - B * fabs(t))) / (kappa_g *

  return y;
}

static double iomega(double x, double z)
{
  double abserr, results;
  int neval;
  PnlFunc func;

  func.F = omegadensity;
  func.params = NULL;
  neval = 50;
  pnl_integration_GK(&func, x, z, 0.0000001, 1, &results, &abserr, &neval);

  return results;
}

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

  y = fabs(t) * exp(A * t - B * fabs(t)) / kappa_g;

  return y;
}

static double sigmabar2(double x, double z)
{
  double abserr, results;
  int neval;
  PnlFunc func;

  func.F = Ldensityx2;
  func.params = NULL;
```

```
  neval = 50;
  pnl_integration_GK(&func, x, z, 0.0000001, 1, &results, &abserr, &neval);

  return results;
}

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


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

  //Lévy measure
  A = theta / SQR(sigma);
  B = sqrt(SQR(theta) + 2 * SQR(sigma) / kappa) / SQR(sigma);
  C = sqrt(2.) / (sigma * sqrt(M_PI));

  //Drift changement for the risk-neutral measure
  omega = theta + log(1 - theta * kappa - SQR(sigma) * kappa / 2.) / kappa;
  //the adjusting for VG term (see carr et al. 1998)
  omegaa = -iomega(-1, 0.0001) - iomega(0.0001, 1) - iomega(1, 100) - iomega(-10

  if (fabs(omega - omegaa) >= 0.001)
    {
      fprintf(stderr, "Stability Condition is not satisfied!\ n");
    }
  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
if (flag_scheme == 1)
  dx = sigma * sqrt(dt);
else
  dx = sqrt((sigmabar2(-0.1, 0) + sigmabar2(0, 0.1)) * 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(x[i]) <= 1)
    emp_mean += proba[i] * x[i];

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

/*Maturity condition*/
dis = exp(-(r - divid + omega) * dt + emp_mean);
exp_drift = exp((r - divid + 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;
}

int CALC(TR_MSS_VG)(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 TreeVG(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
                ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Maturity.Val.V_DATE - ptMo
}

static int CHK_OPT(TR_MSS_VG)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt
    return OK;

  return WRONG;
}
#endif //PremiaCurrentVersion
```

```
PremiaEnumMember schemetreevg_members[] =
{
  { "Improved Scheme", 1 },
  { "MSS Scheme", 2 },
  { NULL, NULLINT }
};

static DEFINE_ENUM(schemetreevg, schemetreevg_members);

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 = &schemetreevg;
      first = 0;
    }

  return OK;
}

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