Help

```
#include  "mer1d_std.h"
#include "pnl/pnl_integration.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_mathtools.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2010+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(TR_MSS_MER)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_MSS_MER)(void *Opt,void *Mod,PricingMethod *
    Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double deltaa_g,mu_g,lambda_g,dt;
//-------------------------------------------------------
    ----
//-------------------------------------------------------
    ----
//-Density Function Merton
//-------------------------------------------------------
    ----

double Ldensity(double t,void *p)
{
  double y;
  double des;
  des=lambda_g/deltaa_g/sqrt(2*M_PI);
  y=des*exp(-SQR(t-mu_g)/(2*SQR(deltaa_g)));

  return y;
}
```

```
static double Levy(double x,double z)
{
  double abserr,results;
  int neval;
  PnlFunc func;
  func.function =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;
  double des;
  des=lambda_g/deltaa_g/sqrt(2*M_PI);
  if(fabs(t)<=1)
  y=(exp(t)-1-t)*des*exp(-SQR(t-mu_g)/(2*SQR(deltaa_g)));
  else
  y=(exp(t)-1)*des*exp(-SQR(t-mu_g)/(2*SQR(deltaa_g)));
  return y;
}

static double iomega(double x,double z)
{
  double abserr,results;
  int neval;
  PnlFunc func;
  func.function =omegadensity;
  func.params = NULL;
  neval=50;
  pnl_integration_GK(&func,x,z,0.0000001,1,&results,&abserr
    ,&neval);

  return results;
}
```

```
static int TreeMer(int am,double S0,NumFunc_1  *p,double T,
    double r,double divid,double sigma,double lambda,double mu,
    double gamma2,int N,double *ptprice,double *ptdelta)
{
  double *P,*stock,*proba,*x,*pr;
  double dx,pu,pd;
  double omega,omegaa,deltaa;
  int i,j,k,N2,N_plus,N_minus,M;
  double exp_drift,dis,emp_mean,sum;

  deltaa=sqrt(gamma2);
  mu_g=mu;
  deltaa_g=deltaa;
  lambda_g=lambda;

  //Drift changement for the risk-neutral measure
  omegaa=SQR(sigma)/2+iomega(-1,0)+iomega(0,1)+iomega(1,100
    )+iomega(-100,-1); //the adjusting for VG term (see carr
    et al. 1998)
  omega=SQR(sigma)/2+lambda*(exp(mu+SQR(deltaa)/2)-1);

   if (fabs(omega-omegaa)>=0.001)
    {
      printf("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));
  pr=(double *)malloc((M+1)*sizeof(double));
  x=(double *)malloc((M+1)*sizeof(double));

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

```
//Space step
dx=sigma*sqrt(dt);

//Modification of Paper MLS
//pu and pd
pu=(exp(r*dt)-exp(-dx))/(exp(dx)-exp(-dx));
pd=1-pu;

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

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

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

pr[M/2]=1.-sum;

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

  sum=0;

  for (i=1;i<=M-1;i++)
     {
        if (i!=M/2)
         {
           proba[i]=pr[i+1]*pd+pr[i-1]*pu;
           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(dis,(double)i)));
          }
      }
```

```
     //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(pr);
  free(x);

  return OK;
}

int CALC(TR_MSS_MER)(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 TreeMer(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S0.Val.V_
    PDOUBLE,
                ptOpt->PayOff.Val.V_NUMFUNC_1,ptOpt->Matu
    rity.Val.V_DATE-ptMod->T.Val.V_DATE,r,divid,ptMod->Sigma.Val
    .V_SPDOUBLE,ptMod->Lambda.Val.V_PDOUBLE,ptMod->Mean.Val.V_
    PDOUBLE,ptMod->Variance.Val.V_PDOUBLE,Met->Par[0].Val.V_INT2
    ,&(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(TR_MSS_MER)(void *Opt, void *Mod)
{
  if ( (strcmp( ((Option*)Opt)->Name,"CallEuro")==0) || (
    strcmp( ((Option*)Opt)->Name,"PutEuro")==0||(strcmp( ((
```

```
    Option*)Opt)->Name,"CallAmer")==0) || (strcmp( ((Option*)Opt)->
    Name,"PutAmer")==0)))
    return OK;

  return WRONG;
}
#endif //PremiaCurrentVersion


static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  static int first=1;

  if (first)
    {
      Met->HelpFilenameHint = "tr_mss_merton";
      Met->Par[0].Val.V_INT2=100;
      first=0;
    }

  return OK;
}

PricingMethod MET(TR_MSS_MER)=
{
  "TR_MSS_MER",
  {{"TimeStepNumber",INT2,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(TR_MSS_MER),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
    ID},{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_MSS_MER),
  CHK_split,
  MET(Init)
};
```

References