Help

```
#include "cgmy1d_pad.h"
#include "enums.h"
#include "pnl/pnl_cdf.h"
#include"pnl/pnl_random.h"
#include"pnl/pnl_specfun.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2011+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(MC_CGMY_FloatingLookback)(void *Opt, voi
    d *Mod)
{
  return NONACTIVE;
}
int CALC(MC_CGMY_FloatingLookback)(void*Opt,void *Mod,Prici
    ngMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else
//Compute the positive or negative jump size between the sm
    allest and the biggest value of cdf_jump_points of the CGMY
    process
static double jump_generator_CGMY(double* cdf_jump_vect,
    double* cdf_jump_points,int cdf_jump_vect_size,double M_G,
    double Y,int generator)
{
   double z,v,y;
   int test,temp,l,j,q;
   test=0;
   v=pnl_rand_uni(generator);
   y=cdf_jump_vect[cdf_jump_vect_size]*v;
   l=cdf_jump_vect_size/2;
   j=cdf_jump_vect_size;
   z=0;
   if(cdf_jump_vect[l]>y)
   {
    l=0;
    j=cdf_jump_vect_size/2;
```

```
   }
   if(v==1)
   {
     z=cdf_jump_points[cdf_jump_vect_size];
   }
   if(v==0)
   {
    z=cdf_jump_points[0];
   }
   if(v!=1 && v!=0)
   {
    while(test==0)
    {
     if(cdf_jump_vect[l+1]>y)
     {
      q=l;
      test=1;
     }
     else
     {
      temp=(j-l-1)/2+l;
      if(cdf_jump_vect[temp]>y)
      {
       j=temp;
       l=l+1;
      }
      else
      {
       l=temp*(temp>l)+(l+1)*(temp<=l);
      }
     }
    }
    z=pow(1/pow(cdf_jump_points[q],Y)-(y-cdf_jump_vect[q])*
    Y*exp(M_G*cdf_jump_points[q]),-1/Y);
   }
 return z;
}
static int CGMY_Mc_FloatingLookback(double s_maxmin,
    NumFunc_2*P,double S0,double T,double r,double divid,double C,
    double G,double M,double Y,int generator,int n_points,long n_
    paths,double *ptprice,double *ptdelta,double *priceerror,
```

```
    double *deltaerror)
{
    double eps,s,s1,s2,s3,s4,s5,s6,sup,inf,infS,supS,payo
    ff,control,proba,discount, drift,err;
    double  sigma,lambda_p,control_expec,lambda_m,cdf_jump_
    bound,pas,min_M_G,*t,cov_payoff_control;
    double var_payoff,var_control,cor_payoff_control,contr
    ol_coef,var_proba,*cdf_jump_points;
    double *cdf_jump_vect_p,*cdf_jump_vect_m,*X,tau,*jump_
    time_vect_p,*jump_time_vect_m,*W,s0,beta;
    int i,j,k,jump_number_p,jump_number_m,m1,m2,cdf_jump_
    vect_size,*N_p,*N_m;
    discount=exp(-r*T);
    err=1E-16;
    eps=0.1;
    beta=0.5826;
    cdf_jump_vect_size=100000;
    X=malloc((n_points+1)*sizeof(double));
    W=malloc((n_points+1)*sizeof(double));
    t=malloc((n_points+1)*sizeof(double));
    N_p=malloc((n_points+1)*sizeof(int));
    N_m=malloc((n_points+1)*sizeof(int));
    X[0]=0;
    W[0]=0;
    t[0]=0;
    pas=T/n_points;
    for(i=1;i<=n_points;i++)
    {
      t[i]=i*pas;
    }
    N_p[0]=0;
    N_m[0]=0;
    control_expec=exp((r-divid)*T);
    s=0;
    s1=0;
    s2=0;
    s3=0;
    s4=0;
    s5=0;
    s6=0;
    if(M<2 || G<=0 || Y>=2 || Y==0)
```

```
{
 printf("Function CGMY_MC_LookbackFloating : invalid
parameters. We must have M>=2, G>0, 0<Y<2{n");
}
lambda_p=C*pow(M,Y)*pnl_sf_gamma_inc(-Y,eps*M);//posi
tive jump intensity
while(lambda_p*T<30)
{
 eps=eps*0.9;
 lambda_p=C*pow(M,Y)*pnl_sf_gamma_inc(-Y,eps*M);
}
lambda_m=C*pow(G,Y)*pnl_sf_gamma_inc(-Y,eps*G);//negat
ive jump intensity
while(lambda_m*T<30)
{
 eps=eps*0.9;
 lambda_m=C*pow(G,Y)*pnl_sf_gamma_inc(-Y,eps*G);
}
lambda_p=C*pow(M,Y)*pnl_sf_gamma_inc(-Y,eps*M);
////////////////////////////////////////////////////////////

cdf_jump_bound=1;
min_M_G=MIN(M,G);
//Computation of the biggest jump that we tolerate
while(C*exp(-min_M_G*cdf_jump_bound)/(min_M_G*pow(cdf_
jump_bound,1+Y))>err)
  cdf_jump_bound++;
pas=(cdf_jump_bound-eps)/cdf_jump_vect_size;
cdf_jump_points=malloc((cdf_jump_vect_size+1)*sizeof(
double));
cdf_jump_vect_p=malloc((cdf_jump_vect_size+1)*sizeof(
double));
cdf_jump_vect_m=malloc((cdf_jump_vect_size+1)*sizeof(
double));
cdf_jump_points[0]=eps;
cdf_jump_vect_p[0]=0;
cdf_jump_vect_m[0]=0;
//computation of the cdf of the positive and negative
jumps at some points
for(i=1;i<=cdf_jump_vect_size;i++)
{
```

```
     cdf_jump_points[i]=i*pas+eps;
     cdf_jump_vect_p[i]=cdf_jump_vect_p[i-1]+exp(-M*cdf_
    jump_points[i-1])*(1/pow(cdf_jump_points[i-1],Y)-1/pow(cdf_
    jump_points[i],Y))/Y;
     cdf_jump_vect_m[i]=cdf_jump_vect_m[i-1]+exp(-G*cdf_
    jump_points[i-1])*(1/pow(cdf_jump_points[i-1],Y)-1/pow(cdf_
    jump_points[i],Y))/Y;
    }
/////////////////////////////////////////////////////
    sigma=sqrt(C*(pow(M,Y-2)*(tgamma(2-Y)-pnl_sf_gamma_inc(
    2-Y,eps*M))+pow(G,Y-2)*(tgamma(2-Y)-pnl_sf_gamma_inc(2-Y,
    eps*G))));
    if(Y==1)
      drift=(r-divid)-C*((M-1)*log(1.-1/M)+(G+1)*log(1.+1/
    G));
    else
      drift=(r-divid)-C*tgamma(-Y)*(pow(M,Y)*(pow(1-1/M,Y)-
    1+Y/M)+pow(G,Y)*(pow(1+1/G,Y)-1-Y/G));
    drift=drift-C*(pow(M,Y-1)*(pnl_sf_gamma_inc(1-Y,eps*M)-
    pnl_sf_gamma_inc(1-Y,M))-pow(G,Y-1)*(pnl_sf_gamma_inc(1-Y,
    eps*G)-pnl_sf_gamma_inc(1-Y,G)));
/////////////////////////////////////////////////////
    m1=(int)(1000*lambda_p*T);
    m2=(int)(1000*lambda_m*T);
    jump_time_vect_p=malloc((m1)*sizeof(double));
    jump_time_vect_m=malloc((m2)*sizeof(double));
    jump_time_vect_p[0]=0;
    jump_time_vect_m[0]=0;
/////////////////////////////////////////////////////
    pnl_rand_init(generator,1,n_paths);
  if ((P->Compute)==&Call_StrikeSpot2)
  {
        s_maxmin=exp(beta*sigma*sqrt(T/n_points))*s_maxmin;
    //shifting the predetermine minimum in order to approxima
    te the continuous price
        for(i=0;i<n_paths;i++)
        {
         //simulation of the positive jump times and number
         tau=-1/(lambda_p)*log(pnl_rand_uni(generator));
         jump_number_p=0;
         while(tau<T)
```

```
        {
         jump_number_p++;
         jump_time_vect_p[jump_number_p]=tau;
         tau+=-1/(lambda_p)*log(pnl_rand_uni(generator));
        }
        //simulation of the negative jump times and number
        tau=-1/(lambda_m)*log(pnl_rand_uni(generator));
        jump_number_m=0;
        while(tau<T)
        {
         jump_number_m++;
         jump_time_vect_m[jump_number_m]=tau;
         tau+=-1/(lambda_m)*log(pnl_rand_uni(generator));
        }
        jump_time_vect_p[jump_number_p+1]=0;
        jump_time_vect_m[jump_number_m+1]=0;

///////////////////////////////////////////////////////////
    ////////////////////////////////
        // simulation of one CGMY path
      for(k=1;k<=n_points;k++)
      {
       W[k]=sigma*pnl_rand_normal(generator)*sqrt(t[k]-t[
    k-1])+drift*(t[k]-t[k-1])+W[k-1];
       N_p[k]=N_p[k-1];
       j=N_p[k-1]+1;
       while(jump_time_vect_p[j]<=t[k] && j<=jump_number_
    p)
       {
        N_p[k]++;
        j++;
       }
       s0=0;
       for(j=N_p[k-1]+1;j<=N_p[k];j++)
         s0+=jump_generator_CGMY(cdf_jump_vect_p,cdf_
    jump_points,cdf_jump_vect_size,M,Y,generator);
       N_m[k]=N_m[k-1];
       j=N_m[k-1]+1;
       while(jump_time_vect_m[j]<=t[k] && j<=jump_number_
    m)
       {
```

```
          N_m[k]++;
          j++;
        }
        for(j=N_m[k-1]+1;j<=N_m[k];j++)
        s0-=jump_generator_CGMY(cdf_jump_vect_m,cdf_jump_
    points,cdf_jump_vect_size,G,Y,generator);
        X[k]=X[k-1]+(W[k]-W[k-1])+s0;
      }
////////////////////////////////////////////////////////////
    ////////////////////////////////
          //computation of the supremum and the infimum of
    the CGMY path
         inf=X[0];
         sup=X[0];
         for(j=1;j<=n_points;j++)
         {
           if(inf>X[j])
             inf=X[j];
           if(sup<X[j])
             sup=X[j];
         }
         proba=0;
         infS=S0*exp(inf);
         if(infS>s_maxmin)
         {
          infS=s_maxmin;
          proba=1;
         }
         payoff=infS;
         infS=S0*exp(X[n_points]-sup);//antithetic variab
    le associated with the exponential of the Levy infimum
         if(infS>s_maxmin)
         {
          infS=s_maxmin;
          proba+=1.;
         }
         proba=proba/2;
         payoff=discount*(payoff+infS)/2;
         control=exp(X[n_points]);
         s+=control;
         s1+=payoff;
```

```
      s2+=payoff*payoff;
      s3+=control*payoff;
      s4+=control*control;
      s5+=proba;
      s6+=proba*proba;
     }
     cov_payoff_control=s3/n_paths-s1*s/((double)n_paths
 *n_paths);
     var_payoff=(s2-s1*s1/((double)n_paths))/((double)n_
 paths-1);
     var_control=(s4-s*s/((double)n_paths))/((double)n_
 paths-1);
     cor_payoff_control=cov_payoff_control/(sqrt(var_pay
 off)*sqrt(var_control));
     control_coef=cov_payoff_control/var_control;
     var_proba=(s6-s5*s5/((double)n_paths))/((double)n_
 paths-1);
     *ptprice=-exp(-beta*sigma*sqrt(T/n_points))*(s1/(
 double)n_paths-control_coef*(s/n_paths-control_expec))+S0*exp(-
 divid*T);
   *priceerror=exp(-beta*sigma*sqrt(T/n_points))*1.96*
 sqrt(var_payoff*(1-cor_payoff_control*cor_payoff_control))/
 sqrt((double)n_paths);
     *ptdelta=(*ptprice+discount*s_maxmin*s5/(double)n_
 paths)/S0;
     *deltaerror=(*priceerror+discount*s_maxmin*1.96*sq
 rt(var_proba)/sqrt((double)n_paths))/S0;
}
else//Put
 if ((P->Compute)==&Put_StrikeSpot2)
 {
s_maxmin=exp(-beta*sigma*sqrt(T/n_points))*s_maxmin;//sh
  ifting the predetermine maximum in order to approximate th
  e continuous price
for(i=0;i<n_paths;i++)
{
 //simulation of the positive jump times and number
 tau=-1/(lambda_p)*log(pnl_rand_uni(generator));
 jump_number_p=0;
 while(tau<T)
 {
```

```
 jump_number_p++;
 jump_time_vect_p[jump_number_p]=tau;
 tau+=-1/(lambda_p)*log(pnl_rand_uni(generator));
}
//simulation of the negative jump times and number
tau=-1/(lambda_m)*log(pnl_rand_uni(generator));
jump_number_m=0;
while(tau<T)
{
 jump_number_m++;
 jump_time_vect_m[jump_number_m]=tau;
 tau+=-1/(lambda_m)*log(pnl_rand_uni(generator));
}
jump_time_vect_p[jump_number_p+1]=0;
jump_time_vect_m[jump_number_m+1]=0;
////////////////////////////////////////////////////////
 ///////////////////////////////////
// simulation of one CGMY path
for(k=1;k<=n_points;k++)
{
 W[k]=sigma*pnl_rand_normal(generator)*sqrt(t[k]-t[k-1]
 )+drift*(t[k]-t[k-1])+W[k-1];
 N_p[k]=N_p[k-1];
 j=N_p[k-1]+1;
 while(jump_time_vect_p[j]<=t[k] && j<=jump_number_p)
 {
  N_p[k]++;
  j++;
 }
 s0=0;
 for(j=N_p[k-1]+1;j<=N_p[k];j++)
   s0+=jump_generator_CGMY(cdf_jump_vect_p,cdf_jump_po
 ints,cdf_jump_vect_size,M,Y,generator);
 N_m[k]=N_m[k-1];
 j=N_m[k-1]+1;
 while(jump_time_vect_m[j]<=t[k] && j<=jump_number_m)
 {
   N_m[k]++;
   j++;
 }
 for(j=N_m[k-1]+1;j<=N_m[k];j++)
```

```
 s0-=jump_generator_CGMY(cdf_jump_vect_m,cdf_jump_po
 ints,cdf_jump_vect_size,G,Y,generator);
 X[k]=X[k-1]+(W[k]-W[k-1])+s0;
}
///////////////////////////////////////////////////////
 ///////////////////////////////////
//computation of the supremum and the infimum of the
 CGMY path
inf=X[0];
sup=X[0];
for(j=1;j<=n_points;j++)
{
  if(inf>X[j])
    inf=X[j];
  if(sup<X[j])
    sup=X[j];
}
proba=0;
supS=S0*exp(sup);
if(supS<s_maxmin)
{
 supS=s_maxmin;
 proba=1.;
}
payoff=supS;
supS=S0*exp(X[n_points]-inf);//antithetic variable ass
 ociated with the exponential of the Levy supremum
if(supS<s_maxmin)
{
 supS=s_maxmin;
 proba+=1.;
}
proba=proba/2;
payoff=discount*(payoff+supS)/2;
control=exp(X[n_points]);
s+=control;
s1+=payoff;
s2+=payoff*payoff;
s3+=control*payoff;
s4+=control*control;
s5+=proba;
```

```
   s6+=proba*proba;
  }
  cov_payoff_control=s3/n_paths-s1*s/((double)n_paths*n_
    paths);
  var_payoff=(s2-s1*s1/((double)n_paths))/(n_paths-1);
  var_control=(s4-s*s/((double)n_paths))/(n_paths-1);
  cor_payoff_control=cov_payoff_control/(sqrt(var_payoff)*
    sqrt(var_control));
  control_coef=cov_payoff_control/var_control;
  var_proba=(s6-s5*s5/((double)n_paths))/(n_paths-1);
  *ptprice=exp(beta*sigma*sqrt(T/n_points))*(s1/n_paths-
    control_coef*(s/n_paths-control_expec))-S0*exp(-divid*T);
  *priceerror=exp(beta*sigma*sqrt(T/n_points))*1.96*sqrt(
    var_payoff*(1-cor_payoff_control*cor_payoff_control))/sqrt(n_
    paths);
  *ptdelta=(*ptprice-discount*s_maxmin*s5/n_paths)/S0;
  *deltaerror=(*priceerror+discount*s_maxmin*1.96*sqrt(
    var_proba)/sqrt((double)n_paths))/S0;
   }
   free(X);
   free(W);
   free(cdf_jump_points);
   free(cdf_jump_vect_p);
   free(cdf_jump_vect_m);
   free(jump_time_vect_p);
   free(jump_time_vect_m);
   free(t);
   free(N_p);
   free(N_m);
  return OK;
}

int CALC(MC_CGMY_FloatingLookback)(void*Opt,void *Mod,Prici
    ngMethod *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  CGMY_Mc_FloatingLookback((ptOpt->PathDep.Val.V_
    NUMFUNC_2)->Par[4].Val.V_PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_2,pt
    Mod->S0.Val.V_PDOUBLE,ptOpt->Maturity.Val.V_DATE-ptMod->T.Val
    .V_DATE,r,divid,ptMod->C.Val.V_PDOUBLE,ptMod->G.Val.V_
    DOUBLE,ptMod->M.Val.V_SPDOUBLE,ptMod->Y.Val.V_PDOUBLE,Met->Par[0
    ].Val.V_ENUM.value,Met->Par[1].Val.V_PINT,Met->Par[2].Val.
    V_LONG,&(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.V_
    DOUBLE),&(Met->Res[2].Val.V_DOUBLE),&(Met->Res[3].Val.V_DOUBLE))
    ;
}

static int CHK_OPT(MC_CGMY_FloatingLookback)(void *Opt, voi
    d *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"    LookBackCallFloatingEuro")==0) || (strcm
    return OK;
  return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Mod)
{
  if ( Met->init == 0)
    {
      Met->init=1;
      Met->HelpFilenameHint = "mc_cgmy_lookbackfloating";
      Met->Par[0].Val.V_ENUM.value=0;
      Met->Par[0].Val.V_ENUM.members=&PremiaEnumMCRNGs;
      Met->Par[1].Val.V_PINT=100;
      Met->Par[2].Val.V_LONG=100000;
    }
  return OK;
}

PricingMethod MET(MC_CGMY_FloatingLookback)=
{
  "MC_CGMY_FloatingLookback",
  {{"RandomGenerator",ENUM,{100},ALLOW},
   {"Number of discretization steps",LONG,{100},ALLOW},{"N
    iterations",LONG,{100},ALLOW},{" ",PREMIA_NULLTYPE,{0},FORBID}
```

```
    },
  CALC(MC_CGMY_FloatingLookback),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
    ID},{"Price Error",DOUBLE,{100},FORBID},{"Delta Error",
    DOUBLE,{100},FORBID},{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_CGMY_FloatingLookback),
  CHK_ok,
  MET(Init)
} ;
```

References