Help

```
#include "kou1d_pad.h"
#include "enums.h"
#include "pnl/pnl_cdf.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2009+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_Kou_Floating)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_Kou_Floating)(void*Opt,void *Mod,PricingMethod
    *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static int Kou_Mc_Floating(double s_maxmin,NumFunc_2*P,
    double S0,double T,double r,double divid,double sigma,double lam
    bda,double lambdap,double lambdam,double p,int generator,i
    nt n_points,long n_paths,double *ptprice,double *ptdelta,
    double *priceerror,double *deltaerror)
{
  double *s,s1,s2,*s3,**s4,s5,s6;
  double pas,sup,inf,supw,infw,exp_sup,exp_inf,exp_supw,exp
    _infw,beta;
  double *t,*jump_size_vect,*jump_time_vect,sup0,temp,nu,
    discount,*cov_payoff_control,var_payoff,**var_control;
  double cor_payoff_control,*coef_control,var_proba,determi
    nant_control,*X,*W,payoff,*control;
  double proba,s0,u0,s_shift,e1,e2,e3,expect_control_0;
  int i,j,k,l,jump_number,*N;

  k=0;
  beta=0.5826;
  nu=((r-divid)-sigma*sigma/2-lambda*(p*lambdap/(lambdap-1)
    +(1-p)*lambdam/(lambdam+1)-1));
  discount=exp(-r*T);
```

```
pas=T/n_points;
X=malloc((n_points+1)*sizeof(double));
W=malloc((n_points+1)*sizeof(double));
t=malloc((n_points+1)*sizeof(double));
N=malloc((n_points+1)*sizeof(int));
s=malloc((2)*sizeof(double));
s3=malloc((2)*sizeof(double));
control=malloc((2)*sizeof(double));
var_control=(double**) malloc((2)*sizeof(double));
s4=(double**) malloc((2)*sizeof(double));
for(i=0;i<=1;i++)
{
 var_control[i]=malloc((2)*sizeof(double));
 s4[i]=malloc((2)*sizeof(double));
}
cov_payoff_control=malloc((2)*sizeof(double));
coef_control=malloc((2)*sizeof(double));
jump_size_vect=malloc((int)(1000*lambda*T)*sizeof(double)
  );
jump_time_vect=malloc((int)(1000*lambda*T)*sizeof(double)
  );
X[0]=0;
W[0]=0;
s[0]=0;
s[1]=0;
s3[0]=0;
s3[1]=0;
s4[0][0]=0;
s4[0][1]=0;
s4[1][0]=0;
s4[1][1]=0;
N[0]=0;
s1=0;
s2=0;
s5=0;
s6=0;
for(k=0;k<=n_points;k++)
{
 t[k]=k*pas;
}
pnl_rand_init(generator,1,n_paths);
```

```
//Call options case
if ((P->Compute)==&Call_StrikeSpot2)
{
      s_shift=exp(beta*sigma*sqrt(T/n_points))*s_maxmin;
      e1=(log(s_maxmin/S0)*(S0>s_maxmin)-(nu+sigma*sigma)
  *T)/(sigma*sqrt(T));
      e2=e1+sigma*sqrt(T);
      e3=(log(s_maxmin/S0)*(S0>s_maxmin)+nu*T)/(sigma*sq
  rt(T));
      //first control variable's expected value : it's th
  e exponential of the infimum of the continuous part of the
  Levy process
      if(S0>=s_maxmin)
        expect_control_0=exp(beta*sigma*sqrt(T/n_points))
  *(discount*s_maxmin+S0*exp((sigma*sigma/2+nu-r)*T)*cdf_nor
  (e1)-s_maxmin*discount*cdf_nor(e2)+S0*discount*(1/(1+2*nu/
  (sigma*sigma)))*(-pow(S0/s_maxmin,-(2*nu/(sigma*sigma)+1))
  *cdf_nor(e3)+exp((nu+sigma*sigma/2)*T)*cdf_nor(e1)));
      else
        expect_control_0=exp(beta*sigma*sqrt(T/n_points))
  *(S0*exp(sigma*sigma/2+nu-r)*(1+1/(2*nu/(sigma*sigma)))*
  cdf_nor(e1)+S0*exp(-r*T)*cdf_nor(e3)/(1+sigma*sigma/(2*nu)))
  ;
      for(i=1;i<=n_paths;i++)
      {
        jump_number=pnl_rand_poisson(lambda*T,generator);
  // number of jumps
        // simulation of the continuous part of the Levy
  process : Brownian motion part
        for(j=1;j<=n_points;j++)
        {
          W[j]=sigma*pnl_rand_normal(generator)*sqrt(pas)
  +nu*pas+W[j-1];
        }
        // simulation of the jump's times and the size of
   the jumps
         jump_size_vect[0]=0;
         jump_time_vect[0]=0;
     for(j=1;j<=jump_number;j++)
 {
   jump_time_vect[j]=pnl_rand_uni_ab(0.,T,generator);
```

```
        u0=pnl_rand_uni(generator);
        if(1-p<=u0)
          jump_size_vect[j]=-log(1-(u0-1+p)/p)/lambdap;
        else
          jump_size_vect[j]=log(u0/(1-p))/lambdam;
}
  //rearranging jump's times in ascending order
for(l=0;l<jump_number;l++)
{
  sup0=jump_time_vect[0];
        k=jump_number-l;
  for(j=0;j<jump_number+1-l;j++)
  {
    if(jump_time_vect[j]>sup0)
    {
       sup0=jump_time_vect[j];
       k=j;
    }
  }
  if(k!=jump_number-l)
  {
     temp=jump_time_vect[jump_number-l];
     jump_time_vect[jump_number-l]=jump_time_vect[k];
     jump_time_vect[k]=temp;
  }
}
  // simulation of one Levy path
for(k=1;k<=n_points;k++)
{
  N[k]=0;
  for(j=1;j<=jump_number;j++)
  {
    if(jump_time_vect[j]<=t[k])
      N[k]++;
  }
  s0=0;
  for(j=N[k-1]+1;j<=N[k];j++)
    s0+=jump_size_vect[j];
  X[k]=X[k-1]+(W[k]-W[k-1])+s0;
}
      //computation of the supremum and the infimum of
```

```
the Levy path and its continuous parts
     inf=X[0];
     sup=X[0];
     infw=W[0];
     supw=W[0];
     for(j=1;j<=n_points;j++)
     {
       if(inf>X[j])
         inf=X[j];
       if(sup<X[j])
         sup=X[j];
       if(infw>W[j])
         infw=W[j];
       if(supw<W[j])
         supw=W[j];
     }
     proba=0;
     exp_inf=S0*exp(inf);
     exp_infw=S0*exp(infw);
     if(exp_inf>s_shift)
     {
      exp_inf=s_shift;
      proba=1.;
     }
     if(exp_infw>s_shift)
     {
      exp_infw=s_shift;
     }
     payoff=exp_inf;
     control[0]=exp_infw;
     exp_inf=S0*exp(X[n_points]-sup);//antithetic
variable associated with the exponential of the Levy infimum
     exp_infw=S0*exp(W[n_points]-supw);//antithetic
variable associated with the first control variate
     if(exp_inf>s_shift)
     {
      exp_inf=s_shift;
      proba+=1.;
     }
     if(exp_infw>s_shift)
     {
```

```
        exp_infw=s_shift;
       }
       proba=proba/2;
       payoff=discount*(payoff+exp_inf)/2;
       control[0]=discount*((control[0]+exp_infw)/2);
       control[1]=discount*S0*exp(X[n_points]);
       s1+=payoff;
       s2+=payoff*payoff;
       for(j=0;j<=1;j++)
       {
        s[j]+=control[j];
        s3[j]+=control[j]*payoff;
        for(k=0;k<=1;k++)
         s4[j][k]+=control[j]*control[k];
       }
       s5+=proba;
       s6+=proba*proba;
      }
      for(j=0;j<=1;j++)
      {
       cov_payoff_control[j]=(s3[j]-s1*s[j]/((double)n_
paths))/(n_paths-1);
       for(k=0;k<=1;k++)
       {
        var_control[j][k]=(s4[j][k]-s[j]*s[k]/((double)n_
paths))/(n_paths-1);
       }
      }
      determinant_control=var_control[0][0]*var_control[1
][1]-var_control[0][1]*var_control[1][0];//determinant of
the var_control matrix
      coef_control[0]=(cov_payoff_control[0]*var_control[
1][1]-cov_payoff_control[1]*var_control[0][1])/determinant
_control;
      coef_control[1]=(cov_payoff_control[1]*var_control[
0][0]-cov_payoff_control[0]*var_control[1][0])/determinant
_control;
      var_payoff=(s2-s1*s1/((double)n_paths))/(n_paths-1)
;
      cor_payoff_control=sqrt((coef_control[0]*cov_payo
ff_control[0]+coef_control[1]*cov_payoff_control[1])/var_pay
```

```
  off);
      var_proba=(s6-s5*s5/((double)n_paths))/(n_paths-1);
      *ptprice=-exp(-beta*sigma*sqrt(T/n_points))*(s1/n_
  paths-coef_control[0]*(s[0]/n_paths-expect_control_0)-coef_
  control[1]*(s[1]/n_paths-S0*exp(-divid*T)))+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*sq
  rt(var_proba)/sqrt((double)n_paths))/S0;
}
else//Put
   if ((P->Compute)==&Put_StrikeSpot2)
   {
        s_shift=exp(-beta*sigma*sqrt(T/n_points))*s_maxmi
  n;
        e1=(log(S0/s_maxmin)*(S0<s_maxmin)+(nu+sigma*si
  gma)*T)/(sigma*sqrt(T));
        e2=e1-sigma*sqrt(T);
        e3=(log(S0/s_maxmin)*(S0<s_maxmin)-nu*T)/(sigma*
  sqrt(T));
        //first control variable's expected value : it's
  the exponential of the supremum of the continuous part of
  the Levy process
        if(S0<=s_maxmin)
          expect_control_0=exp(-beta*sigma*sqrt(T/n_po
  ints))*(discount*s_maxmin+S0*exp((sigma*sigma/2+nu-r)*T)*cdf_
  nor(e1)-s_maxmin*discount*cdf_nor(e2)+ S0*discount*(1/(1+2*
  nu/(sigma*sigma)))*(-pow(S0/s_maxmin,-(2*nu/(sigma*sigma)+1)
  )*cdf_nor(e3)+exp((nu+sigma*sigma/2)*T)*cdf_nor(e1)));
        else
          expect_control_0=exp(-beta*sigma*sqrt(T/n_po
  ints))*(S0*exp(sigma*sigma/2+nu-r)*(1+1/(2*nu/(sigma*sigma)))
  *cdf_nor(e1)+S0*exp(-r*T)*cdf_nor(e3)/(1+sigma*sigma/(2*
  nu)))+S0*exp(-divid*T);
        for(i=0;i<n_paths;i++)
        {
         jump_number=pnl_rand_poisson(lambda*T,    generator);// number of jum
         //simulation of the continuous part of the Levy
```

```
: Brownian motion part
       for(j=1;j<=n_points;j++)
       {
        W[j]=sigma*pnl_rand_normal(generator)*sqrt(pas)
+nu*pas+W[j-1];
       }
       jump_size_vect[0]=0;
       jump_time_vect[0]=0;
       // simulation of the jump's times and the size
of the jumps
 for(j=1;j<=jump_number;j++)
 {
   jump_time_vect[j]=pnl_rand_uni_ab(0.,T,generator);
         u0=pnl_rand_uni(generator);
         if(1-p<=u0)
           jump_size_vect[j]=-log(1-(u0-1+p)/p)/lambd
ap;
         else
           jump_size_vect[j]=log(u0/(1-p))/lambdam;
 }
       //rearranging jump's times in ascending order
   for(l=0;l<jump_number;l++)
 {
   sup0=jump_time_vect[0];
         k=jump_number-l;
   for(j=0;j<jump_number+1-l;j++)
   {
      if(jump_time_vect[j]>sup0)
      {
        sup0=jump_time_vect[j];
        k=j;
      }
   }
   if(k!=jump_number-l)
   {
     temp=jump_time_vect[jump_number-l];
     jump_time_vect[jump_number-l]=jump_time_vect[k];
           jump_time_vect[k]=temp;
   }
       }
       // simulation of one Levy path
```

```
for(k=1;k<=n_points;k++)
{
  N[k]=0;
  for(j=1;j<=jump_number;j++)
  {
     if(jump_time_vect[j]<=t[k])
     N[k]++;
  }
  s0=0;
  for(j=N[k-1]+1;j<=N[k];j++)
    s0+=jump_size_vect[j];
  X[k]=X[k-1]+(W[k]-W[k-1])+s0;
}
      //computation of the supremum and the infimum of
the Levy path and its continuous parts
      sup=X[0];
      inf=X[0];
      infw=W[0];
      supw=W[0];
      for(j=1;j<=n_points;j++)
      {
        if(inf>X[j])
          inf=X[j];
        if(sup<X[j])
          sup=X[j];
        if(infw>W[j])
          infw=W[j];
        if(supw<W[j])
          supw=W[j];
      }
      proba=0;
      exp_sup=S0*exp(sup);
      exp_supw=S0*exp(supw);
      if(exp_sup<s_shift)
      {
       exp_sup=s_shift;
       proba=1.;
      }
      if(exp_supw<s_shift)
      {
       exp_supw=s_shift;
```

```
        }
        payoff=exp_sup;
        control[0]=exp_supw;
        exp_sup=S0*exp(X[n_points]-inf);//antithetic
variable associated with the exponential of the Levy supremum
        exp_supw=S0*exp(W[n_points]-infw);//antithetic
variable associated with the first control variable
        if(exp_sup<s_shift)
        {
         exp_sup=s_shift;
         proba+=1.;
        }
        if(exp_supw<s_shift)
        {
         exp_supw=s_shift;
        }
        proba=proba/2.;
        payoff=discount*(payoff+exp_sup)/2;
        control[0]=discount*(control[0]+exp_supw)/2;
        control[1]=discount*S0*exp(X[n_points]);
        s1+=payoff;
        s2+=payoff*payoff;
        for(j=0;j<=1;j++)
        {
          s[j]+=control[j];
          s3[j]+=control[j]*payoff;
          for(k=0;k<=1;k++)
            s4[j][k]+=control[j]*control[k];
        }
        s5+=proba;
        s6+=proba*proba;
       }
       for(j=0;j<=1;j++)
       {
         cov_payoff_control[j]=(s3[j]-s1*s[j]/((double)
n_paths))/(n_paths-1);
         for(k=0;k<=1;k++)
         {
           var_control[j][k]=(s4[j][k]-s[j]*s[k]/((
double)n_paths))/(n_paths-1);
         }
```

```
      }
      determinant_control=var_control[0][0]*var_contro
l[1][1]-var_control[0][1]*var_control[1][0];//determinant
of the var_control matrix
      coef_control[0]=(cov_payoff_control[0]*var_contro
l[1][1]-cov_payoff_control[1]*var_control[0][1])/determina
nt_control;
      coef_control[1]=(cov_payoff_control[1]*var_contro
l[0][0]-cov_payoff_control[0]*var_control[1][0])/determina
nt_control;
      var_payoff=(s2-s1*s1/((double)n_paths))/(n_paths-
1);
      cor_payoff_control=sqrt((coef_control[0]*cov_payo
ff_control[0]+coef_control[1]*cov_payoff_control[1])/var_pay
off);
      var_proba=(s6-s5*s5/((double)n_paths))/(n_paths-1
);
      *ptprice=exp(beta*sigma*sqrt(T/n_points))*(s1/n_
paths-coef_control[0]*(s[0]/n_paths-expect_control_0)-coef_
control[1]*(s[1]/n_paths-S0*exp(-divid*T)))-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(t);
  free(N);
  free(coef_control);
  free(control);
  free(s);
  free(s3);
  free(cov_payoff_control);
  for(i=0;i<=1;i++)
  {
   free(var_control[i]);
   free(s4[i]);
```

```
      }
      free(var_control);
      free(s4);
      free(jump_time_vect);
      free(jump_size_vect);
     return OK;
}

int CALC(MC_Kou_Floating)(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  Kou_Mc_Floating((ptOpt->PathDep.Val.V_NUMFUNC_2)-
    >Par[4].Val.V_PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_2,ptMod-
    >S0.Val.V_PDOUBLE,ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.
    V_DATE,r,divid,ptMod->Sigma.Val.V_PDOUBLE,ptMod->Lambda.Val
    .V_PDOUBLE,ptMod->LambdaPlus.Val.V_PDOUBLE,ptMod->LambdaM
    inus.Val.V_PDOUBLE,ptMod->P.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_Kou_Floating)(void *Opt, void *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->Par[0].Val.V_ENUM.value=0;
      Met->Par[0].Val.V_ENUM.members=&PremiaEnumMCRNGs;
      Met->Par[1].Val.V_PINT=25;
      Met->Par[2].Val.V_LONG=100000;
    }
  return OK;
}

PricingMethod MET(MC_Kou_Floating)=
{
  "MC_Kou_LookbackFloating",
  {{"RandomGenerator",ENUM,{100},ALLOW},
   {"Number of discretization steps",LONG,{100},ALLOW},{"N
    iterations",LONG,{100},ALLOW},{" ",PREMIA_NULLTYPE,{0},FORBID}
    },
  CALC(MC_Kou_Floating),
  {{"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_Kou_Floating),
  CHK_ok,
  MET(Init)
} ;
```

References