Help
```
extern "C"{
#include "kou1d_lim.h"
#include "enums.h"
}
#include"pnl/pnl_random.h"
#include "pnl/pnl_cdf.h"
#include"pnl/pnl_mathtools.h"
#include"pnl/pnl_root.h"

extern "C"{

#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_Kou_Out_LRM)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_Kou_Out_LRM)(void*Opt,void *Mod,PricingMethod *
    Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else
//Algorithme de tri croissant
static void tri_up(double* x, int size)
{
 double sup,temp;
 int i,j,k=0;
 for(i=0;i<size-1;i++)
 {
  sup=x[0];
  for(j=0;j<size-i;j++)
  {
   if(x[j]>sup)
   {
    sup=x[j];
    k=j;
   }
  }
```

```
  if(k!=size-i-1)
  {
   temp=x[size-i-1];
   x[size-i-1]=x[k];
   x[k]=temp;
  }
 }
}

int Kou_Mc_Out_Lrm(int b_type,double l,double rebate,
    double S0,NumFunc_1  *P,double
T,double r,double divid,double sigma,double lambda,double
    lambdap,double
lambdam,double p,int generator,int n_points,int n_paths,
    double *ptPrice,double *ptDelta,double *priceError,double *delt
    aError)
{
    double sum_payoffScoreFunction,sum_square_payoffScore
    Function,payoff,log_l_S0,sum_payoff;
    double nu,u0,*jump_time_vect,discount,var_payoffScore
    Function,*X,*W,K;
    double *jump_size_vect,scoreFunction,beta,*t,s0,inf,su
    m_scoreFunction,sum_square_payoff;
    double cov_payoff_payoffScoreFunction,var_payoff,sum_
    payoff_payoffScoreFunction,sup;
    int i,j,k,jump_number,*N,n_vect;
    nu=((r-divid)-sigma*sigma/2-lambda*(p*lambdap/(lambdap-
    1)+(1-p)*lambdam/(lambdam+1)-1));
    K=P->Par[0].Val.V_DOUBLE;
    beta=0.5826;
    discount=exp(-r*T);
    log_l_S0=log(l/S0);
    sum_payoffScoreFunction=0;
    sum_square_payoffScoreFunction=0;
    sum_scoreFunction=0;
    sum_payoff=0;
    sum_payoff_payoffScoreFunction=0;
    sum_square_payoff=0;
    t=(double *)malloc((n_points+1)*sizeof(double));
    for(i=0;i<=n_points;i++)
     t[i]=i*T/n_points;
```

```
N=(int *)malloc((n_points+1)*sizeof(int));
N[0]=0;
X=(double *)malloc((n_points+1)*sizeof(double));
W=(double *)malloc((n_points+1)*sizeof(double));
W[0]=0;
X[0]=0;
n_vect=intapprox(1000*lambda*T);
jump_size_vect=(double *)malloc(n_vect*sizeof(double));
jump_time_vect=(double *)malloc(n_vect*sizeof(double));
pnl_rand_init(generator,1,n_paths);
/*Down Case*/
   if(b_type==0)
   {
    if((P->Compute)==&Call)//call
    {
     for(i=0;i<n_paths;i++)
     {
      jump_number=pnl_rand_poisson(lambda*T,generator);
      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;
      }
      jump_time_vect[jump_number+1]=T;
      jump_size_vect[jump_number+1]=0;
    tri_up(jump_time_vect,jump_number+1);//rearranging
 jump's times in ascending order
   // simulation of the Brownian motion part at jump'
s times
      for(j=1;j<=n_points;j++)
      {
        W[j]=sigma*pnl_rand_normal(generator)*sqrt(t[j]
-t[j-1])+nu*(t[j]-t[j-1])+W[j-1];
      }
      // simulation of one Levy process X at jump's
times
```

```
      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;
    }
      inf=X[0];
      for(j=1;j<=n_points;j++)
      {
        if(inf>X[j])
          inf=X[j];
      }
      payoff=discount*(S0*exp(X[n_points])-K)*(S0*exp(X
[n_points])>K)*(inf-beta*sigma*sqrt(T/n_points)>log_l_S0)+
rebate*(inf-beta*sigma*sqrt(T/n_points)<=log_l_S0);
      scoreFunction=(W[1]-nu*t[1])/(sigma*sigma*t[1]*S0
);
      sum_payoff+=payoff;
      sum_scoreFunction+=scoreFunction;
      sum_square_payoff+=payoff*payoff;
      sum_payoffScoreFunction+=payoff*scoreFunction;
      sum_square_payoffScoreFunction+=payoff*scoreFunct
ion*payoff*scoreFunction;
      sum_payoff_payoffScoreFunction+=payoff*scoreFunct
ion*payoff;
     }
     var_payoff=(sum_square_payoff-sum_payoff*sum_payo
ff/n_paths)/(n_paths-1);
     var_payoffScoreFunction=(sum_square_payoffScore
Function-sum_payoffScoreFunction*sum_payoffScoreFunction/n_paths)/
(n_paths-1);
     cov_payoff_payoffScoreFunction=(sum_payoff_payoffS
coreFunction-sum_payoff*sum_payoffScoreFunction/n_paths)/(n_
paths-1);
```

```
      *ptPrice=sum_payoff/n_paths;
      *priceError=1.96*sqrt(var_payoff)/sqrt((double)n_
paths);
      *ptDelta=sum_payoffScoreFunction/n_paths-(sum_scor
eFunction/((double)n_paths))*(sum_payoff/((double)n_paths))
;
      *deltaError=1.96*sqrt(var_payoffScoreFunction+
var_payoff*(sum_scoreFunction/n_paths)*(sum_scoreFunction/n_
paths)-2*(sum_scoreFunction/n_paths)*cov_payoff_payoffScore
Function)/sqrt((double)n_paths);
     }
     if((P->Compute)==&Put)//put
     {
      for(i=0;i<n_paths;i++)
      {
       jump_number=pnl_rand_poisson(lambda*T,generator);
       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;
       }
       jump_time_vect[jump_number+1]=T;
       jump_size_vect[jump_number+1]=0;
     tri_up(jump_time_vect,jump_number+1);//rearranging
 jump's times in ascending order
    // simulation of the Brownian motion part at jump'
s times
       for(j=1;j<=n_points;j++)
       {
         W[j]=sigma*pnl_rand_normal(generator)*sqrt(t[j]
-t[j-1])+nu*(t[j]-t[j-1])+W[j-1];
       }
       // simulation of one Levy process X at jump's
times
       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;
    }
      inf=X[0];
      for(j=1;j<=n_points;j++)
      {
        if(inf>X[j])
          inf=X[j];
      }
      payoff=discount*(K-S0*exp(X[n_points]))*(S0*exp(X
[n_points])<K)*(inf-beta*sigma*sqrt(T/n_points)>log_l_S0)+
rebate*(inf-beta*sigma*sqrt(T/n_points)<=log_l_S0);
      scoreFunction=(W[1]-nu*t[1])/(sigma*sigma*t[1]*S0
);
      sum_payoff+=payoff;
      sum_scoreFunction+=scoreFunction;
      sum_square_payoff+=payoff*payoff;
      sum_payoffScoreFunction+=payoff*scoreFunction;
      sum_square_payoffScoreFunction+=payoff*scoreFunct
ion*payoff*scoreFunction;
      sum_payoff_payoffScoreFunction+=payoff*scoreFunct
ion*payoff;
     }
     var_payoff=(sum_square_payoff-sum_payoff*sum_payo
ff/n_paths)/(n_paths-1);
     var_payoffScoreFunction=(sum_square_payoffScore
Function-sum_payoffScoreFunction*sum_payoffScoreFunction/n_paths)/
(n_paths-1);
     cov_payoff_payoffScoreFunction=(sum_payoff_payoffS
coreFunction-sum_payoff*sum_payoffScoreFunction/n_paths)/(n_
paths-1);
```

```
     *ptPrice=sum_payoff/n_paths;
     *priceError=1.96*sqrt(var_payoff)/sqrt((double)n_
paths);
     *ptDelta=sum_payoffScoreFunction/n_paths-(sum_scor
eFunction/((double)n_paths))*(sum_payoff/((double)n_paths))
;
     *deltaError=1.96*sqrt(var_payoffScoreFunction+
var_payoff*(sum_scoreFunction/n_paths)*(sum_scoreFunction/n_
paths)-2*(sum_scoreFunction/n_paths)*cov_payoff_payoffScore
Function)/sqrt((double)n_paths);
    }
   }
/*Up Case*/
   if(b_type==1)
   {
    if((P->Compute)==&Call)//call
    {
     for(i=0;i<n_paths;i++)
     {
      jump_number=pnl_rand_poisson(lambda*T,generator);
      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;
      }
      jump_time_vect[jump_number+1]=T;
      jump_size_vect[jump_number+1]=0;
    tri_up(jump_time_vect,jump_number+1);//rearranging
 jump's times in ascending order
   // simulation of the Brownian motion part at jump'
s times
      for(j=1;j<=n_points;j++)
      {
        W[j]=sigma*pnl_rand_normal(generator)*sqrt(t[j]
-t[j-1])+nu*(t[j]-t[j-1])+W[j-1];
      }
```

```
      // simulation of one Levy process X at jump's
times
      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;
    }
      sup=X[0];
      for(j=1;j<=n_points;j++)
      {
        if(sup<X[j])
          sup=X[j];
      }
      payoff=discount*(S0*exp(X[n_points])-K)*(S0*exp(X
[n_points])>K)*(sup+beta*sigma*sqrt(T/n_points)<log_l_S0)+
rebate*(sup+beta*sigma*sqrt(T/n_points)>=log_l_S0);
      scoreFunction=(W[1]-nu*t[1])/(sigma*sigma*t[1]*S0
);
      sum_payoff+=payoff;
      sum_scoreFunction+=scoreFunction;
      sum_square_payoff+=payoff*payoff;
      sum_payoffScoreFunction+=payoff*scoreFunction;
      sum_square_payoffScoreFunction+=payoff*scoreFunct
ion*payoff*scoreFunction;
      sum_payoff_payoffScoreFunction+=payoff*scoreFunct
ion*payoff;
     }
     var_payoff=(sum_square_payoff-sum_payoff*sum_payo
ff/n_paths)/(n_paths-1);
     var_payoffScoreFunction=(sum_square_payoffScore
Function-sum_payoffScoreFunction*sum_payoffScoreFunction/n_paths)/
(n_paths-1);
     cov_payoff_payoffScoreFunction=(sum_payoff_payoffS
```

```
coreFunction-sum_payoff*sum_payoffScoreFunction/n_paths)/(n_
paths-1);

     *ptPrice=sum_payoff/n_paths;
     *priceError=1.96*sqrt(var_payoff)/sqrt((double)n_
paths);
     *ptDelta=sum_payoffScoreFunction/n_paths-(sum_scor
eFunction/((double)n_paths))*(sum_payoff/((double)n_paths))
;
     *deltaError=1.96*sqrt(var_payoffScoreFunction+
var_payoff*(sum_scoreFunction/n_paths)*(sum_scoreFunction/n_
paths)-2*(sum_scoreFunction/n_paths)*cov_payoff_payoffScore
Function)/sqrt((double)n_paths);
    }
    if((P->Compute)==&Put)//put
    {
     for(i=0;i<n_paths;i++)
     {
      jump_number=pnl_rand_poisson(lambda*T,generator);
      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;
      }
      jump_time_vect[jump_number+1]=T;
      jump_size_vect[jump_number+1]=0;
    tri_up(jump_time_vect,jump_number+1);//rearranging
 jump's times in ascending order
   // simulation of the Brownian motion part at jump'
s times
      for(j=1;j<=n_points;j++)
      {
        W[j]=sigma*pnl_rand_normal(generator)*sqrt(t[j]
-t[j-1])+nu*(t[j]-t[j-1])+W[j-1];
      }
      // simulation of one Levy process X at jump's
```

```
times
      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;
    }
      sup=X[0];
      for(j=1;j<=n_points;j++)
      {
        if(sup<X[j])
          sup=X[j];
      }
      payoff=discount*(K-S0*exp(X[n_points]))*(S0*exp(X
[n_points])<K)*(sup+beta*sigma*sqrt(T/n_points)<log_l_S0)+
rebate*(sup+beta*sigma*sqrt(T/n_points)>=log_l_S0);
      scoreFunction=(W[1]-nu*t[1])/(sigma*sigma*t[1]*S0
);
      sum_payoff+=payoff;
      sum_scoreFunction+=scoreFunction;
      sum_square_payoff+=payoff*payoff;
      sum_payoffScoreFunction+=payoff*scoreFunction;
      sum_square_payoffScoreFunction+=payoff*scoreFunct
ion*payoff*scoreFunction;
      sum_payoff_payoffScoreFunction+=payoff*scoreFunct
ion*payoff;
     }
     var_payoff=(sum_square_payoff-sum_payoff*sum_payo
ff/n_paths)/(n_paths-1);
     var_payoffScoreFunction=(sum_square_payoffScore
Function-sum_payoffScoreFunction*sum_payoffScoreFunction/n_paths)/
(n_paths-1);
     cov_payoff_payoffScoreFunction=(sum_payoff_payoffS
coreFunction-sum_payoff*sum_payoffScoreFunction/n_paths)/(n_
```

```
    paths-1);

         *ptPrice=sum_payoff/n_paths;
         *priceError=1.96*sqrt(var_payoff)/sqrt((double)n_
    paths);
         *ptDelta=sum_payoffScoreFunction/n_paths-(sum_scor
    eFunction/((double)n_paths))*(sum_payoff/((double)n_paths))
    ;
         *deltaError=1.96*sqrt(var_payoffScoreFunction+
    var_payoff*(sum_scoreFunction/n_paths)*(sum_scoreFunction/n_
    paths)-2*(sum_scoreFunction/n_paths)*cov_payoff_payoffScore
    Function)/sqrt((double)n_paths);
        }
       }
       free(jump_time_vect);
       free(jump_size_vect);
       free(X);
       free(W);
       free(N);
       free(t);
   return OK;
}
int CALC(MC_Kou_Out_LRM)(void*Opt,void *Mod,PricingMethod *
    Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    double r,divid,limit,rebate;
    int upordown;

    r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
    divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
    limit=((ptOpt->Limit.Val.V_NUMFUNC_1)->Compute)((ptOpt-
    >Limit.Val.V_NUMFUNC_1)->Par,ptMod->T.Val.V_DATE);
    rebate=((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compute)((pt
    Opt->Rebate.Val.V_NUMFUNC_1)->Par,ptMod->T.Val.V_DATE);

    if ((ptOpt->DownOrUp).Val.V_BOOL==DOWN)
      upordown=0;
    else upordown=1;
```

```
    return  Kou_Mc_Out_Lrm(upordown,limit,rebate,ptMod->S0.
    Val.V_PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_1,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->LambdaMinus.Val.V_PDOUBLE,ptMod->P.Val.V_PDO
    UBLE,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_Out_LRM)(void *Opt, void *Mod)
{
    Option* ptOpt=(Option*)Opt;
    TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

    if ((opt->OutOrIn).Val.V_BOOL==OUT)
      if ((opt->EuOrAm).Val.V_BOOL==EURO)
        if ((opt->Parisian).Val.V_BOOL==WRONG)
          return  OK;

    return WRONG;
}
#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
    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=50;
      Met->Par[2].Val.V_LONG=100000;
    }
    return OK;
}
PricingMethod MET(MC_Kou_Out_LRM)=
{
    "MC_Kou_Out_LRM",
    {
        {"RandomGenerator",ENUM,{100},ALLOW},
```

```
        {"Number of discretization steps",LONG,{100},ALLOW}
    ,
        {"N iterations",LONG,{100},ALLOW},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(MC_Kou_Out_LRM),
    {
        {"Price",DOUBLE,{100},FORBID},
        {"Delta",DOUBLE,{100},FORBID},
        {"Price Error",DOUBLE,{100},FORBID},
        {"Delta Error",DOUBLE,{100},FORBID},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CHK_OPT(MC_Kou_Out_LRM),
    CHK_ok,
    MET(Init)
} ;
}
```

References