```
  Help
#include<stdlib.h>
#include<math.h>
#include"pnl/pnl_random.h"
#include"pnl/pnl_specfun.h"
#include"pnl/pnl_mathtools.h"
#include "nig1d_pad.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_NIG_FixedAsian)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_NIG_FixedAsian)(void*Opt,void *Mod,Pricing
    Method *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 NIG
    process
static double jump_generator_NIG(double* cdf_jump_vect,
    double* cdf_jump_points,int cdf_jump_vect_size,double alpha,
    double beta,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./cdf_jump_points[q]-(y-cdf_jump_vect[q])*exp(-
    beta*cdf_jump_points[q])/(pnl_bessel_k(1.,alpha*cdf_jump_po
    ints[q])*cdf_jump_points[q]),-1.);
   }
 return z;
}

//(exp(x)-1)/x
static double p_func(double x)
{
```

```
 double s;
 int i,n;
 n=1;
 s=0;
 for(i=0;i<=n;i++)
   s+=pow(x,i)/pnl_fact(i+1);

 return s;
}

//(4exp(x)+(2x-3)exp(2x)-1)/x^3
static double var_func(double x)
{
 double s;
 int i,n;
 n=1;
 s=0;
 for(i=0;i<=n;i++)
   s+=4*pow(x,i)/pnl_fact(i+3)-3*pow(2.,i+3)*pow(x,i)/pnl_
    fact(i+3)+pow(2.,i+3)*pow(x,i)/pnl_fact(i+2);

 return s;
}

//exp(x)/x-(exp(x)-1)/x^2
static double cov_func(double x)
{
 double s;
 int i,n;
 n=1;
 s=0;
 for(i=0;i<=n;i++)
   s+=pow(x,i)*(1./pnl_fact(i+1)-1./pnl_fact(i+2));

 return s;
}

static int NIG_Mc_FixedAsian(NumFunc_2*P,double S0,double
T,double r,double divid,double sigma,double theta,double ka
    ppa,int generator,int n_paths,double *ptprice,double *ptde
    lta,double *priceerror,double *deltaerror)
```

```
{
    double eps,s,s1,s2,s3,s4,s5,s6,payoff,dpayoff,control,
    discount,w1,w2,drift,err,u,u0,z,sigma0;
    double lambda_p,control_expec,lambda_m,cdf_jump_bound,
    pas,cov_payoff_control,var_payoff,var_control;
    double cor_payoff_control,control_coef,var_dpayoff,*cdf
    _jump_points,*cdf_jump_vect_p;
    double *cdf_jump_vect_m,*Xg,*Xd,tau,*jump_time_vect,*
    jump_time_vect_p,*jump_time_vect_m;
    double alpha,beta,delta,var_temp,cov_temp,*vect_temp,g_
    temp,K;
    int i,j,k,jump_number_p,jump_number_m,jump_number,m1,m2
    ,cdf_jump_vect_size,n_int,k1,k2;
    K=P->Par[0].Val.V_DOUBLE;
    n_int=10000;
    discount=exp(-r*T);
    err=1E-16;
    eps=0.1;
    cdf_jump_vect_size=100000;
    if(r-divid!=0)
     control_expec=S0*(exp((r-divid)*T)-1)/((r-divid)*T);
    else
     control_expec=S0;
    s=0;
    s1=0;
    s2=0;
    s3=0;
    s4=0;
    s5=0;
    s6=0;
    alpha=sqrt(theta*theta+sigma*sigma/kappa)/(sigma*sigma)
    ;
    beta=theta/(sigma*sigma);
    delta=sigma/sqrt(kappa);
    if(alpha-fabs(beta)<1)
    {
     printf("Function NIG_Mc_FixedAsian: invalid paramete
    rs. We must have sqrt(kappa)*(2*fabs(theta)+sigma*sigma)<=1{
    n");
    }
    while(delta*exp(-fabs(beta)*eps)/(M_PI*eps)<10)
```

```
      eps=eps*0.9;
////////////////////////////////////////////////////////////

    cdf_jump_bound=1;
    //Computation of the biggest jump that we tolerate
    while(2*sqrt(alpha/(2*M_PI))*delta*exp(-(alpha-fabs(bet
    a))*cdf_jump_bound)/((alpha-fabs(beta))*pow(cdf_jump_bound,
    1.5))>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(beta*cdf_
    jump_points[i-1])*cdf_jump_points[i-1]*pnl_bessel_k(1.,alpha*
    cdf_jump_points[i-1])*(1/cdf_jump_points[i-1]-1/cdf_jump_po
    ints[i]);
     cdf_jump_vect_m[i]=cdf_jump_vect_m[i-1]+exp(-beta*cdf_
    jump_points[i-1])*cdf_jump_points[i-1]*pnl_bessel_k(1.,alpha*
    cdf_jump_points[i-1])*(1/cdf_jump_points[i-1]-1/cdf_jump_po
    ints[i]);
    }
    lambda_p=cdf_jump_vect_p[cdf_jump_vect_size]*alpha*delt
    a/M_PI;
    lambda_m=cdf_jump_vect_m[cdf_jump_vect_size]*alpha*delt
    a/M_PI;
/////////////////////////////////////////////////////
    sigma0=0;
    for(i=1;i<=n_int;i++)
     sigma0+=(eps*i/n_int)*cosh(beta*i*eps/n_int)*pnl_bess
```

```
    el_k(1.,alpha*i*eps/n_int)*eps/n_int;
    sigma0=sqrt(sigma0*alpha*delta*2/M_PI);
    drift=0;
    for(i=1;i<=n_int;i++)
     drift+=sinh(beta*i*eps/n_int)*pnl_bessel_k(1.,alpha*i*
    eps/n_int)*eps/n_int;
   drift=drift*alpha*delta*2/M_PI+(r-divid)-delta*(sqrt(alp
    ha*alpha-beta*beta)-sqrt(alpha*alpha-(beta+1)*(beta+1)));
////////////////////////////////////////////////////
    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=malloc((m1+m2)*sizeof(double));
    vect_temp=malloc((m1+m2)*sizeof(double));
    jump_time_vect_p[0]=0;
    jump_time_vect_m[0]=0;
    jump_time_vect[0]=0;
    vect_temp[0]=0;
    Xg=malloc((m1+m2)*sizeof(double));//left value of X at
    jump times
    Xd=malloc((m1+m2)*sizeof(double));//right value of X
    at jump times
    Xg[0]=0;
    Xd[0]=0;
////////////////////////////////////////////////////
    pnl_rand_init(generator,1,n_paths);
    /*Call Case*/
    if((P->Compute)==&Call_OverSpot2)
    {
       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 numb
    er
        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]=T;
        jump_time_vect_m[jump_number_m+1]=T;
        jump_number=jump_number_p+jump_number_m;
////////////////////////////////////////////////////////////
    //
        //computation of Xg and Xd
      k1=1;
      k2=1;
      u0=0;
        u=0;
      for(k=1;k<=jump_number;k++)
      {
       w1=jump_time_vect_p[k1];
       w2=jump_time_vect_m[k2];
       if(w1<w2)
       {
        u=w1;
        k1++;
          z=jump_generator_NIG(cdf_jump_vect_p,cdf_jump_po
    ints,cdf_jump_vect_size,alpha,beta,generator);
       }
       else
       {
        u=w2;
        k2++;
        z=-jump_generator_NIG(cdf_jump_vect_m,cdf_jump_po
    ints,cdf_jump_vect_size,alpha,-beta,generator);
       }
         g_temp=pnl_rand_normal(generator);
         if(fabs(drift*(u-u0))<1e-4)
         {
```

```
        var_temp=(u-u0)*(u-u0)*(u-u0)*var_func(drift*(u-
u0))/2;
        cov_temp=(u-u0)*(u-u0)*cov_func(drift*(u-u0));

       }
       else
       {
        var_temp=(4*exp(drift*(u-u0))+(2*drift*(u-u0)-3)
*exp(2*drift*(u-u0))-1)/(2*drift*drift*drift);
        cov_temp=(u-u0)*exp(drift*(u-u0))/drift-(exp(dr
ift*(u-u0))-1)/(drift*drift);
       }
     jump_time_vect[k]=u;
       vect_temp[k]=cov_temp*g_temp/(sqrt(u-u0))+sqrt(
var_temp-cov_temp*cov_temp/(u-u0))*pnl_rand_normal(generator);
     Xg[k]=drift*(u-u0)+sigma0*g_temp*sqrt(u-u0)+Xd[k-1
];
     Xd[k]=Xg[k]+z;
     u0=u;
    }
      g_temp=pnl_rand_normal(generator);
      if(fabs(drift*(T-u0))<1e-4)
      {
       var_temp=(T-u0)*(T-u0)*(T-u0)*var_func(drift*(T-
u0))/2;
       cov_temp=(T-u0)*(T-u0)*cov_func(drift*(T-u0));
      }
      else
      {
       var_temp=(4*exp(drift*(T-u0))+(2*drift*(T-u0)-3)*
exp(2*drift*(T-u0))-1)/(2*drift*drift*drift);
       cov_temp=(T-u0)*exp(drift*(T-u0))/drift-(exp(drif
t*(T-u0))-1)/(drift*drift);
      }
    jump_time_vect[jump_number+1]=T;
      vect_temp[jump_number+1]=cov_temp*g_temp/(sqrt(T-
u0))+sqrt(var_temp-cov_temp*cov_temp/(T-u0))*pnl_rand_nor
mal(generator);
    Xg[jump_number+1]=drift*(T-u0)+sigma0*g_temp*sqrt(
T-u0)+Xd[jump_number];
    Xd[jump_number+1]=Xg[jump_number+1];
```

```
///////////////////////////////////////////////////////////
    /
          //computation of the payoff
          payoff=0;
          for(j=1;j<=jump_number+1;j++)
          {
           if(fabs(drift*(jump_time_vect[j]-jump_time_vect[
    j-1]))<1e-4)
            payoff+=exp(Xd[j-1])*(p_func(drift*(jump_time_v
    ect[j]-jump_time_vect[j-1]))*(jump_time_vect[j]-jump_time_v
    ect[j-1])+sigma0*vect_temp[j]);
           else
            payoff+=exp(Xd[j-1])*((exp(drift*(jump_time_vec
    t[j]-jump_time_vect[j-1]))-1)/drift+sigma0*vect_temp[j]);
          }
          control=S0*payoff/T;

          dpayoff=-discount*(payoff/T)*(S0*payoff/T<K);
          payoff=discount*(K-S0*payoff/T)*(S0*payoff/T<K);

          s1+=payoff;
          s+=payoff*payoff;
          s2+=control;
          s3+=control*control;
          s4+=control*payoff;
          s5+=dpayoff;
          s6+=dpayoff*dpayoff;
        }
        cov_payoff_control=s4/n_paths-s1*s2/((double)n_
    paths*n_paths);
        var_payoff=(s-s1*s1/((double)n_paths))/(n_paths-1);
        var_control=(s3-s2*s2/((double)n_paths))/(n_paths-1
    );
        cor_payoff_control=cov_payoff_control/(sqrt(var_pay
    off)*sqrt(var_control));
        control_coef=cov_payoff_control/var_control;
        var_dpayoff=(s6-s5*s5/((double)n_paths))/(n_paths-1
    );
        if(r!=divid)
         *ptprice=(s1/n_paths-control_coef*(s2/n_paths-
    control_expec))-K*exp(-r*T)+S0*(exp(-divid*T)-exp(-r*T))/((r-
```

```
 divid)*T);
     else
      *ptprice=(s1/n_paths-control_coef*(s2/n_paths-
 control_expec))+(S0-K)*exp(-r*T);
     *priceerror=1.96*sqrt(var_payoff*(1-cor_payoff_
 control*cor_payoff_control))/sqrt(n_paths);
     if(r!=divid)
      *ptdelta=s5/(n_paths)+(exp(-divid*T)-exp(-r*T))/((
 r-divid)*T);
     else
      *ptdelta=s5/(n_paths)+exp(-r*T);
     *deltaerror=1.96*sqrt(var_dpayoff)/sqrt(n_paths);
}
/*Put case*/
 if((P->Compute)==&Put_OverSpot2)
 {
    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 numb
 er
      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]=T;
      jump_time_vect_m[jump_number_m+1]=T;
      jump_number=jump_number_p+jump_number_m;
////////////////////////////////////////////////////////////
```

```
//
     //computation of Xg and Xd
   k1=1;
   k2=1;
   u0=0;
     u=0;
   for(k=1;k<=jump_number;k++)
   {
    w1=jump_time_vect_p[k1];
    w2=jump_time_vect_m[k2];
    if(w1<w2)
    {
     u=w1;
     k1++;
       z=jump_generator_NIG(cdf_jump_vect_p,cdf_jump_po
ints,cdf_jump_vect_size,alpha,beta,generator);
    }
    else
    {
     u=w2;
     k2++;
     z=-jump_generator_NIG(cdf_jump_vect_m,cdf_jump_po
ints,cdf_jump_vect_size,alpha,-beta,generator);
    }
      g_temp=pnl_rand_normal(generator);
      if(fabs(drift*(u-u0))<1e-4)
      {
       var_temp=(u-u0)*(u-u0)*(u-u0)*var_func(drift*(u-
u0))/2;
       cov_temp=(u-u0)*(u-u0)*cov_func(drift*(u-u0));

      }
      else
      {
       var_temp=(4*exp(drift*(u-u0))+(2*drift*(u-u0)-3)
*exp(2*drift*(u-u0))-1)/(2*drift*drift*drift);
       cov_temp=(u-u0)*exp(drift*(u-u0))/drift-(exp(dr
ift*(u-u0))-1)/(drift*drift);
      }
    jump_time_vect[k]=u;
      vect_temp[k]=cov_temp*g_temp/(sqrt(u-u0))+sqrt(
```

```
var_temp-cov_temp*cov_temp/(u-u0))*pnl_rand_normal(generator);
    Xg[k]=drift*(u-u0)+sigma0*g_temp*sqrt(u-u0)+Xd[k-1
];
    Xd[k]=Xg[k]+z;
    u0=u;
   }
     g_temp=pnl_rand_normal(generator);
     if(fabs(drift*(T-u0))<1e-4)
     {
      var_temp=(T-u0)*(T-u0)*(T-u0)*var_func(drift*(T-
u0))/2;
      cov_temp=(T-u0)*(T-u0)*cov_func(drift*(T-u0));
     }
     else
     {
      var_temp=(4*exp(drift*(T-u0))+(2*drift*(T-u0)-3)*
exp(2*drift*(T-u0))-1)/(2*drift*drift*drift);
      cov_temp=(T-u0)*exp(drift*(T-u0))/drift-(exp(drif
t*(T-u0))-1)/(drift*drift);
     }
   jump_time_vect[jump_number+1]=T;
     vect_temp[jump_number+1]=cov_temp*g_temp/(sqrt(T-
u0))+sqrt(var_temp-cov_temp*cov_temp/(T-u0))*pnl_rand_nor
mal(generator);
   Xg[jump_number+1]=drift*(T-u0)+sigma0*g_temp*sqrt(
T-u0)+Xd[jump_number];
   Xd[jump_number+1]=Xg[jump_number+1];
////////////////////////////////////////////////////////////
/
     //computation of the payoff
     payoff=0;
     for(j=1;j<=jump_number+1;j++)
     {
      if(fabs(drift*(jump_time_vect[j]-jump_time_vect[
j-1]))<1e-4)
       payoff+=exp(Xd[j-1])*(p_func(drift*(jump_time_v
ect[j]-jump_time_vect[j-1]))*(jump_time_vect[j]-jump_time_v
ect[j-1])+sigma0*vect_temp[j]);
      else
       payoff+=exp(Xd[j-1])*((exp(drift*(jump_time_vec
t[j]-jump_time_vect[j-1]))-1)/drift+sigma0*vect_temp[j]);
```

```
    }
    control=S0*payoff/T;

    dpayoff=-discount*(payoff/T)*(S0*payoff/T<K);
    payoff=discount*(K-S0*payoff/T)*(S0*payoff/T<K);

    s1+=payoff;
    s+=payoff*payoff;
    s2+=control;
    s3+=control*control;
    s4+=control*payoff;
    s5+=dpayoff;
    s6+=dpayoff*dpayoff;
   }
   cov_payoff_control=s4/n_paths-s1*s2/((double)n_
paths*n_paths);
   var_payoff=(s-s1*s1/((double)n_paths))/(n_paths-1);
   var_control=(s3-s2*s2/((double)n_paths))/(n_paths-1
);
   cor_payoff_control=cov_payoff_control/(sqrt(var_pay
off)*sqrt(var_control));
   control_coef=cov_payoff_control/var_control;
   var_dpayoff=(s6-s5*s5/((double)n_paths))/(n_paths-1
);
   *ptprice=(s1/n_paths-control_coef*(s2/n_paths-contr
ol_expec));
   *priceerror=1.96*sqrt(var_payoff*(1-cor_payoff_
control*cor_payoff_control))/sqrt(n_paths);
   *ptdelta=s5/(n_paths);
   *deltaerror=1.96*sqrt(var_dpayoff)/sqrt(n_paths);

  }
  free(Xd);
  free(Xg);
  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(jump_time_vect);
  free(vect_temp);
```

```
   return OK;
}
int CALC(MC_NIG_FixedAsian)(void*Opt,void *Mod,Pricing
    Method *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  NIG_Mc_FixedAsian(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->Thet
    a.Val.V_DOUBLE,ptMod->Kappa.Val.V_SPDOUBLE,Met->Par[0].Val.
    V_ENUM.value,Met->Par[1].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_NIG_FixedAsian)(void *Opt, void *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"AsianCallFixedEuro")==0
    ) || (strcmp( ((Option*)Opt)->Name,"AsianPutFixedEuro")==0
    ) )
    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_LONG=100000;
    }
```

```
  return OK;
}

PricingMethod MET(MC_NIG_FixedAsian)=
{
  "MC_NIG_AsianFixed",
  {{"RandomGenerator",ENUM,{100},ALLOW}, {"N iterations",
    LONG,{100},ALLOW},{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_NIG_FixedAsian),
  {{"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_NIG_FixedAsian),
  CHK_ok,
  MET(Init)
} ;
```

References