

## Help

```

#include "nig1d_pad.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_specfun.h"
// #include "pnl_bessel.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_mathtools.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
    (2011+2) //The "#else" part of the code will be freely available after the (year of creation of this file + 2)
static int CHK_OPT(MC_Nig_Fixed)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_Nig_Fixed)(void*Opt,void *Mod,PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else
//Compute the positive or negative jump size between the smallest 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+1;
            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;
}

```

```

static int NIG_Mc_Fixed(double s_maxmin, NumFunc_2*P, double
    S0, double T, double r, double divid, double sigma, double thet
    a, double kappa, int generator, int n_points, long n_paths,
    double *ptprice, double *ptdelta, double *priceerror, double *delt
    aerror)
{
    double K;
    double eps, s, s1, s2, s3, s4, s5, s6, sup, inf, infS, supS, payo
    ff, control, proba, discount, drift, err;
    double sigma0, lambda_p, control_expec, lambda_m, cdf_jump_
    bound, pas, *t, cov_payoff_control, var_payoff;
    double var_control, cor_payoff_control, control_coef,
    var_proba, *cdf_jump_points, *cdf_jump_vect_p;
    double *cdf_jump_vect_m, *X, tau, *jump_time_vect_p, *jump_
    time_vect_m, *W, s0, alpha, beta, delta, beta1;
    int i, j, k, jump_number_p, jump_number_m, m1, m2, cdf_jump_
    vect_size, *N_p, *N_m, n_int;
    K=P->Par[0].Val.V_DOUBLE;
    n_int=10000;
    discount=exp(-r*T);
    err=1E-16;
    eps=0.1;
    beta1=0;//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;
    pas=T/n_points;
    for(i=0; 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;
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_Fixed: invalid parameters. We
must have sqrt(kappa)*(2*fabs(theta)+sigma*sigma)<=1{n");
}
while(delta*exp(-fabs(beta)*eps)/(M_PI*eps)<17)
    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(beta)
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]*pn1_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_p[0]=0;
jump_time_vect_m[0]=0;
////////////////////////////////////
pnl_rand_init(generator,1,n_paths);
if ((P->Compute)==&Call_OverSpot2)
{
s_maxmin=MAX(s_maxmin,K);
s_maxmin=exp(-beta1*sigma0*sqrt(T/n_points))*s_max
min;//shifting the predetermine maximum in order to approx
imate 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 NIG path
    for(k=1;k<=n_points;k++)
    {
        W[k]=sigma0*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_NIG(cdf_jump_vect_p,cdf_jump_points,cdf_jump_vect_size,alpha,beta,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_NIG(cdf_jump_vect_m,cdf_jump_points
,cdf_jump_vect_size,alpha,-beta,generator);
X[k]=X[k-1]+(W[k]-W[k-1])+s0;
}
////////////////////////////////////
////////////////////////////////////
    //computation of the supremum and the infimum of
the NIG 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 variab
le associated 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_pay
off)*sqrt(var_control));
    control_coef=cov_payoff_control/var_control;
    var_proba=(s6-s5*s5/((double)n_paths))/(n_paths-1);

    *ptprice=exp(beta1*sigma0*sqrt(T/n_points))*(s1/n_
paths-control_coef*(s/n_paths-control_expec))-K*exp(-r*T);
    *priceerror=exp(beta1*sigma0*sqrt(T/n_points))*1.96
*sqrt(var_payoff*(1-cor_payoff_control*cor_payoff_control)
)/sqrt(n_paths);
    *ptdelta=(*ptprice+(K*exp(-r*T)-S0*exp(-divid*T))-
discount*s_maxmin*s5/n_paths)/S0+exp(-divid*T);
    *deltaerror=(*priceerror+discount*s_maxmin*1.96*sq
rt(var_proba)/sqrt((double)n_paths))/S0;
}
else//Put
if ((P->Compute)==&Put_OverSpot2)
{
    s_maxmin=MIN(s_maxmin,K);
    s_maxmin=exp(beta1*sigma0*sqrt(T/n_points))*s_maxmi
n;//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
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]=0;
        jump_time_vect_m[jump_number_m+1]=0;

////////////////////////////////////
////////////////////////////////////
        // simulation of one NIG path
for(k=1;k<=n_points;k++)
{
    W[k]=sigma0*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_NIG(cdf_jump_vect_p,cdf_jump_points,cdf_jump_vect_size,alpha,beta,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_NIG(cdf_jump_vect_m,cdf_jump_points
    ,cdf_jump_vect_size,alpha,-beta,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(-beta1*sigma0*sqrt(T/n_points))*(s1/(
double)n_paths-control_coef*(s/n_paths-control_expec))+K*exp(-r*
T);
*priceerror=exp(-beta1*sigma0*sqrt(T/n_points))*1.96*sq
rt(var_payoff*(1-cor_payoff_control*cor_payoff_control))/sq
rt((double)n_paths);
    *ptdelta=(*ptprice+(S0*exp(-divid*T)-K*exp(-r*T))+
discount*s_maxmin*s5/(double)n_paths)/S0-exp(-divid*T);
    *deltaerror=(*priceerror+discount*s_maxmin*1.96*sq
rt(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_Nig_Fixed)(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 NIG_Mc_Fixed((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_DA
        TE,r,divid,ptMod->Sigma.Val.V_PDOUBLE,ptMod->Theta.Val.V_
        DOUBLE,ptMod->Kappa.Val.V_SPDOUBLE,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_Nig_Fixed)(void *Opt, void *Mod)
{
    if ((strcmp(((Option*)Opt)->Name,"LookBackCallFixedEuro")
        ==0) || (strcmp( ((Option*)Opt)->Name,"    LookBackPutFixedEuro")==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_PINT=100;
        Met->Par[2].Val.V_LONG=100000;
    }
}

```

```

    return OK;
}

PricingMethod MET(MC_Nig_Fixed)=
{
    "MC_NIG_LookbackFixed",
    {"RandomGenerator",ENUM,{100},ALLOW},
    {"Number of discretization steps",LONG,{100},ALLOW},{ "N
iterations",LONG,{100},ALLOW},{ " ",PREMIA_NULLTYPE,{0},FORBID}
    },
    CALC(MC_Nig_Fixed),
    {"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_Fixed),
    CHK_ok,
    MET(Init)
} ;

```

## References