

## Help

```

#include "variancegamma1d_pad.h"
#include "enums.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_specfun.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
    (2010+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_VarianceGamma_Fixed)(void *Opt, void
    *Mod)
{
    return NONACTIVE;
}
int CALC(MC_VarianceGamma_Fixed)(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 VG
    process
static double jump_generator_VG(double* cdf_jump_vect,
    double* cdf_jump_points,int cdf_jump_vect_size,double M_G,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=cdf_jump_points[q]*exp((y-cdf_jump_vect[q])*exp(M_G*
        cdf_jump_points[q]));
    }
    return z;
}

static int VG_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,long n_paths,double *ptprice,
    double *ptdelta,double *errorprice,double *errordelta)
{
    double K;

```

```

double payoff,s,s1,sup,inf,eps,err,*Xg,*Xd,*jump_time_v
ect_p,*jump_time_vect_m;
double proba,lambda_p,lambda_m,cdf_jump_bound,drift,
control,s2,s3,s4,s5,s6;
double u,u0,w1,w2,z,C,G,M,control_expec,cov_payoff_
control,var_payoff,var_control;
double cor_payoff_control,control_coef, tau,pas,*cdf_
jump_points,*cdf_jump_vect_p;
double *cdf_jump_vect_m,min_M_G,var_proba,infS,supS;
int i,j,jump_number,jump_number_p,jump_number_m,cdf_
jump_vect_size,m1,m2,k1,k2,k;
G=sqrt(2/kappa+theta*theta/(sigma*sigma))/sigma+theta/(
sigma*sigma);
M=sqrt(2/kappa+theta*theta/(sigma*sigma))/sigma-theta/(
sigma*sigma);
C=1/kappa;
control_expec=exp((r-divid)*T)*S0;
err=1e-16;
eps=1e-1;
cdf_jump_vect_size=100000;
s=0;
s1=0;
s2=0;
s3=0;
s4=0;
s5=0;
s6=0;
lambda_p=0;
lambda_m=0;
proba=0;
K=P->Par[0].Val.V_DOUBLE;
//////////////////////////////////////
//////////////////////////////////////
lambda_p=C*pnl_sf_gamma_inc(0.,eps*M);//positive jump
intensity
while(lambda_p*T<20)
{
    eps=eps*0.9;
    lambda_p=C*pnl_sf_gamma_inc(0.,eps*M);
}
lambda_m=C*pnl_sf_gamma_inc(0.,eps*G);//negative jump intensity

```

```

while(lambda_m*T<20)
{
    eps=eps*0.9;
    lambda_m=C*pnl_sf_gamma_inc(0.,eps*G);
}
lambda_p=C*pnl_sf_gamma_inc(0.,eps*M);
drift=(r-divid)+log(1-(theta+sigma*sigma/2)*kappa)/kappa
a+theta-C*(exp(-M)/M-exp(-G)/G)-C*((exp(-M*eps)-exp(-M))/M-
(exp(-G*eps)-exp(-G))/G);
////////////////////////////////////
////////////////////////////////////
m1=(int)(10*lambda_p*T);
m2=(int)(10*lambda_m*T);
jump_time_vect_p=malloc((m1)*sizeof(double));
jump_time_vect_m=malloc((m2)*sizeof(double));
Xd=malloc((m2+m1)*sizeof(double));//right value of X
at jump times
Xg=malloc((m2+m1)*sizeof(double));//left value of X at
jump times
jump_time_vect_p[0]=0;
jump_time_vect_m[0]=0;
Xd[0]=0;
Xg[0]=0;
cdf_jump_bound=5;
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*cdf_jump_
bound)>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])*log(cdf_jump_points[i]/cdf_jump_points[i-1])
;
    cdf_jump_vect_m[i]=cdf_jump_vect_m[i-1]+exp(-G*cdf_
jump_points[i-1])*log(cdf_jump_points[i]/cdf_jump_points[i-1])
;
}
//////////////////////////////////////
/
pnl_rand_init(generator,1,n_paths);
//Call options case
if ((P->Compute)==&Call_OverSpot2)
{
    s_maxmin=MAX(s_maxmin,K);
    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));
        }
        jump_time_vect_p[jump_number_p+1]=T;
        //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_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;
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_VG(cdf_jump_vect_p,cdf_jump_points,
cdf_jump_vect_size,M,generator);
}
else
{
u=w2;
k2++;
z=-jump_generator_VG(cdf_jump_vect_m,cdf_jump_points
,cdf_jump_vect_size,G,generator);
}
Xg[k]=drift*(u-u0)+Xd[k-1];
Xd[k]=Xg[k]+z;
u0=u;
}
Xg[jump_number+1]=drift*(T-u0)+Xd[jump_number];
Xd[jump_number+1]=Xg[jump_number+1];
////////////////////////////////////
/
//computation of the supremum and the infimum of the
Levy path
inf=0;
sup=0;
for(j=1;j<=jump_number;j++)
{
if(drift>0)
{
if(inf>Xd[j])

```

```

        inf=Xd[j];
        if(sup<Xg[j])
            sup=Xg[j];
    }
    else
    {
        if(inf>Xg[j])
            inf=Xg[j];
        if(sup<Xd[j])
            sup=Xd[j];
    }
}
supS=S0*exp(sup);
if(supS<s_maxmin)
{
    supS=s_maxmin;
    proba=1;
}
payoff=supS;
supS=S0*exp(Xd[jump_number+1]-inf);//antithetic variat
e associated with the exponential of the Levy supremum
if(supS<s_maxmin)
{
    supS=s_maxmin;
    proba+=1;
}
payoff=(payoff+supS)/2;
proba/=2;
s1+=payoff;
s+=payoff*payoff;
control=S0*exp(Xd[jump_number+1]);
s2+=control;
s3+=control*control;
s4+=control*payoff;
s5+=proba;
s6+=proba*proba;
}
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_payoff)
*sqrt(var_control));
control_coef=cov_payoff_control/var_control;
var_proba=(s6-s5*s5/((double)n_paths))/(n_paths-1);
*ptprice=exp(-r*T)*(s1/n_paths-control_coef*(s2/n_paths
-control_expec))-K*exp(-r*T);
*errorprice=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))-exp(-
r*T)*s_maxmin*s5/(n_paths))/S0+exp(-divid*T);
*errordelta=(*errorprice+1.96*exp(-r*T)*s_maxmin*sqrt(
var_proba)/sqrt(n_paths))/S0;
}
else//Put
if ((P->Compute)==&Put_OverSpot2)
{
s_maxmin=MIN(s_maxmin,K);
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));
}
jump_time_vect_p[jump_number_p+1]=T;
//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_m[jump_number_m+1]=T;
jump_number=jump_number_p+jump_number_m;//total jump
number

```



```

////////////////////////////////////
//
k1=1;
k2=1;
u0=0;
//computation of Xg and Xd
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_VG(cdf_jump_vect_p,cdf_jump_points,
cdf_jump_vect_size,M,generator);
}
else
{
u=w2;
k2++;
z=-jump_generator_VG(cdf_jump_vect_m,cdf_jump_points
,cdf_jump_vect_size,G,generator);
}
Xg[k]=drift*(u-u0)+Xd[k-1];
Xd[k]=Xg[k]+z;
u0=u;
}
Xg[jump_number+1]=drift*(T-u0)+Xd[jump_number];
Xd[jump_number+1]=Xg[jump_number+1];
////////////////////////////////////
/
//computation of the supremum and the infimum of the
Levy path
inf=0;
sup=0;
for(j=1;j<=jump_number;j++)
{
if(drift>0)
{
if(inf>Xd[j])

```

```

        inf=Xd[j];
        if(sup<Xg[j])
            sup=Xg[j];
    }
    else
    {
        if(inf>Xg[j])
            inf=Xg[j];
        if(sup<Xd[j])
            sup=Xd[j];
    }
}
infS=S0*exp(inf);
if(infS>s_maxmin)
{
    infS=s_maxmin;
    proba=1;
}
payoff=infS;
infS=S0*exp(Xd[jump_number+1]-sup); //antithetic variat
e associated with the exponential of the Levy infimum
if(infS>s_maxmin)
{
    infS=s_maxmin;
    proba+=1;
}
payoff=(payoff+infS)/2;
proba/=2;
s1+=payoff;
s+=payoff*payoff;
control=S0*exp(Xd[jump_number+1]);
s2+=control;
s3+=control*control;
s4+=control*payoff;
s5+=proba;
s6+=proba*proba;
}
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_payoff)
    *sqrt(var_control));
    control_coef=cov_payoff_control/var_control;
    var_proba=(s6-s5*s5/((double)n_paths))/(n_paths-1);
    *ptprice=K*exp(-r*T)-(exp(-r*T)*s1/n_paths-control_coe
    f*(s2/n_paths-control_expec));
    *errorprice=1.96*sqrt(var_payoff*(1-cor_payoff_control*
    cor_payoff_control))/sqrt(n_paths);
    *ptdelta=(*ptprice+(S0*exp(-divid*T)-K*exp(-r*T))+exp(-
    r*T)*s_maxmin*s5/(n_paths))/S0-exp(-divid*T);
    *errordelta=(*errorprice+1.96*exp(-r*T)*s_maxmin*sqrt(
    var_proba)/sqrt(n_paths))/S0;
}
free(jump_time_vect_p);
free(jump_time_vect_m);
free(cdf_jump_vect_p);
free(cdf_jump_vect_m);
free(cdf_jump_points);
free(Xd);
free(Xg);
return OK;
}

int CALC(MC_VarianceGamma_Fixed)(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 VG_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_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_VarianceGamma_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_LONG=10000;
    }
    return OK;
}

PricingMethod MET(MC_VarianceGamma_Fixed)=
{
    "MC_VG_LookbackFixed",
    {{ "RandomGenerator",ENUM,{100},ALLOW},
    {"N iterations",LONG,{100},ALLOW},{ " ",PREMIA_NULLTYPE,{0
    },FORBID}},
    CALC(MC_VarianceGamma_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_VarianceGamma_Fixed),
    CHK_ok,
    MET(Init)
} ;

```

## References