

[Help](#)

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#else

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <time.h>

#include "pnl/pnl_complex.h"
#include "pnl/pnl_mathtools.h"
#include "nrutil.h"
#include "../moments.h"

/*-----*/
dcomplex cfGauss(double sg, dcomplex g)
{
//(** characteristic functions**)
//GAUSSIAN CASE

    dcomplex charexp;

    charexp = RCmul(-sg * sg / 2, Cmul(g, g));
    return Cexp(charexp);
}

//NIG
dcomplex cfNig(double alpha, double beta, double delta, dcomplex g)
{
    dcomplex cimm;

    dcomplex charexp;
    dcomplex term1;
```

```

double term2;
dcomplex Ig;

cimm = Complex(0, 1);          /// 'cimm = i

Ig = Cmul(cimm, g);
//'(Sqrt[alpha^2 - (beta + I*g)^2] - term2)
//printf("a=%.9f b=%.9f \ t %.9f \ n", alpha, beta);
term2 = POW((alpha * alpha - beta * beta), 0.5);
//printf("term2=%.9f \ t \ n", term2);
term1 = Cadd(Complex(beta, 0), Ig);
//printf("term1=%.9f \ t %.9f \ n", Creal(term1), Cimag(term1));
term1 = Csqrt(Csub(Complex(alpha * alpha, 0), Cmul(term1, term1)));
//printf("term1=%.9f \ t %.9f \ n", Creal(term1), Cimag(term1));
term1 = Csub(term1, Complex(term2, 0));
//printf("term1=%.9f \ t %.9f \ n", Creal(term1), Cimag(term1));
charexp = RCmul(-delta, term1);

//printf("charexp=%.9f \ t %.9f \ n", Creal(Cexp(charexp)), Cimag(Cexp(charexp))
return Cexp(charexp);
}

//'meixner
dcomplex cfMeixner(double alpha, double beta, double delta, dcomplex g)
{
//'(Cos[beta/2]/Cosh[(alpha*g - I*beta)/2])^(2*delta)

double num;
dcomplex term1;
dcomplex charexp;

num = cos(beta / 2);

term1 = RCmul(alpha / 2, g);
term1 = Csub(term1, Complex(0, beta / 2));
term1 = Ccosh(term1);
term1 = Cdiv(Complex(num, 0), term1);
term1 = Clog(term1);
charexp = RCmul(2 * delta, term1);

```

```

    return Cexp(charexp);
}

```

```

dcomplex cfVarianceGamma(double sg, double nu, double theta, dcomplex g)
{

```

```

    dcomplex term1, term2, root;

```

```

    dcomplex charexp;

```

```

    term1 = RCmul(0.5 * sg * sg * nu, Cmul(g, g)); // '0.5*sg*sg*

```

```

    term1 = Cadd(Complex(1, 0), term1); // '1+0.5*sg*sg*nu*g*g

```

```

    term2 = Cmul(Complex(0, -theta * nu), g); // '- I*theta*g*nu

```

```

    root = Cadd(term1, term2);

```

```

    charexp = RCmul(-1 / nu, Clog(root));

```

```

// '(-1/nu)*log(1 + 0.5*sg*sg*nu*g*g- I*theta*g*nu )

```

```

    return Cexp(charexp);
}

```

```

//Function cfVarianceGammaCGM(Cvg As Double, Gvg As Double, Mvg As Double, g As

```

```

//Dim num As Double

```

```

//Dim den As Variant

```

```

//Dim term1, term2, term3, ratio As Variant

```

```

//Dim charexp As Variant

```

```

//Dim pg As Double

```

```

//pg = 3.1415926536

```

```

//num = Gvg * Mvg //GM

```

```

//term1 = Complex(Gvg * Mvg, 0) //GM

```

```

//term2 = Cmul(Complex(0, (Mvg - Gvg)), g) // (M-G)*I*g

```

```

//term3 = Cmul(g, g) //g^2

```

```

//den = Cadd(Cadd(term1, term2), term3)
//ratio = Cdiv(Complex(num, 0), den)
//charexp = RCmul(Cvg, Clog(ratio))

'G M +(M-G)*I*g+g^2
'R=GM/(G M +(M-G)*I*g+g^2)

//'(1 - I*theta*g*nu + 0.5*sg*sg*nu*g*g)^(-1/nu)
//cfVarianceGammaCGM = Cexp(charexp)

//End Function

//'cgmy
dcomplex cfCgmy(double ccc, double ggg, double mmm, double yyy, dcomplex g)
{
    dcomplex cimm;
    dcomplex charexp;
    dcomplex term1;
    dcomplex term2;
    dcomplex term3;
    dcomplex term4;
    dcomplex cccc;
    dcomplex cggg;
    dcomplex cmmm;
    dcomplex cyyy;

    dcomplex Ig;
    cimm = Complex(0 , 1);
    Ig = Cmul(cimm, g);

    cyyy = Complex(yyy, 0);
    cmmm = Complex(mmm, 0);
    cggg = Complex(ggg, 0);
    cccc = Complex(ccc, 0);

    //'(mmm - I*g)^yyy
    term1 = Cpow(Csub(cmmm, Ig), cyyy);

    //'mmm^yyy
    term2 = Cpow(cmmm, cyyy);

    //'(ggg + I*g)^yyy

```

```

    term3 = Cpow(Cadd(cggg, Ig), cyyy);

    //'ggg^yyy
    term4 = Cpow(cggg, cyyy);

    charexp = Cmul(cccc, Cmul(Ctgamma(RCmul(-1, cyyy)), Csub(Cadd(Csub(term1, term
    return Cexp(charexp);
}

```

```

    //'de

```

```

dcomplex cfDe(double sg, double lambda, double p, double eta1, double eta2, dcomplex
{
    dcomplex dterm;
    double nterm;
    dcomplex term1;
    dcomplex term2;
    dcomplex charexp;

    dterm = Cadd(Complex(eta2, 0), Cmul(Complex(0, 1), g));
    nterm = (1 - p) * eta2;
    term1 = Cdiv(Complex(nterm, 0), dterm);
    nterm = p * eta1;
    dterm = Csub(Complex(eta1, 0), Cmul(Complex(0, 1), g));
    term2 = Cdiv(Complex(nterm, 0), dterm);
    charexp = RCmul(lambda, Csub(Cadd(term1, term2), Complex(1, 0)));
    charexp = Cadd(RCmul(-sg * sg / 2, Cmul(g, g)), charexp);

    return Cexp(charexp);
}

```

```

    ///'merton

```

```

dcomplex cfMerton(double sg, double alpha, double lambda, double delta, dcomplex
{
    dcomplex term1;
    dcomplex charexp;

```

```

    term1 = RCmul(-delta * delta / 2, Cmul(g, g));
    term1 = Csub(Cexp(Cadd(Cmul(Complex(0, alpha), g), term1)), Complex(1, 0));
    term1 = RCmul(lambda, term1);

    charexp = Cadd(RCmul(-sg * sg / 2, Cmul(g, g)), term1);

    return Cexp(charexp);
}

///

```

```

Ig = Cmul(cimm, g); //i*g
Ig = RCmul(-2, Ig); //-2*i*g
term1 = Cadd(Complex(b * b, 0), Ig); //-2*i*g+b^2
term1 = Cpow(term1, Complex(0.5, 0.0)); //sqrt(-2*i*g+b^2)

term2 = Csub(term1, Complex(b, 0.0)); //sqrt(-2*i*g+b^2)-b
charexp = RCmul(-a, term2);

//printf("charexp =%.9f \ t %.9f \ n", Creal(Cexp(charexp)), Cimag(Cexp(charexp))
return Cexp(charexp);
}

```

```

dcomplex cfLevy(int model, double dt, dcomplex g, double parameters[])
{
    dcomplex cf;

    if (model == 1)
    {
        cf = cfGauss(parameters[1] * sqrt(dt), g);
    }

    if (model == 2)
    {
        cf = cfNig(parameters[ 1], parameters[ 2], parameters[ 3] * dt, g);
    }

    if (model == 3)
    {
        cf = cfMeixner(parameters[ 1], parameters[ 2], parameters[ 3] * dt, g);
    }

    if (model == 4)
    {
        cf = cfVarianceGamma(parameters[ 1] * sqrt(dt), parameters[ 2] / dt, param

    }

    if (model == 5)

```

```

    {
        cf = cfCgmy(parameters[ 1] * dt, parameters[ 2], parameters[ 3], parameter
    }

    if (model == 6)
    {
        cf = cfDe(parameters[ 1] * sqrt(dt), parameters[ 2] * dt, parameters[ 3],
    }

    if (model == 7)
    {
        cf = cfMerton(parameters[ 1] * sqrt(dt), parameters[ 2], parameters[ 3] *
    }

    if (model == 8)
    {
        cf = cfShiftedGamma(parameters[ 1] * dt, parameters[ 2], g);
    }

    if (model == 9)
    {
        cf = cfShiftedIG(parameters[ 1] * dt, parameters[ 2] , g);
    }

    return cf;
}

dcomplex cfrn(int model, double rf, double dt, dcomplex g, double parameters[])
{
    double m;
    dcomplex mdtg;
    dcomplex result;

    m = Creal(Csub(Complex(rf, 0), Clog(cfLevy(model, 1, Complex(0, -1), parameter
    mdtg = Cmul(Complex(0, m * dt), g);
    result = Cmul(Cexp(mdtg), cfLevy(model, dt, g, parameters));
    return result;
}

```



```

dcomplex cfrncall(int model, double rf, double dt, dcomplex g, double aa, double parameters[])
{
    dcomplex term1, term2, cfcall;

    ///'(aa^2 + aa - g^2 + I*(2*aa + 1)*g)
    term1 = cfrn(model, rf, dt, Csub(g, Complex(0, (aa + 1))), parameters);

    term2 = Csub(Complex(aa * aa + aa, 0), Cmul(g, g));

    term2 = Cadd(term2, Cmul(Complex(0, (2 * aa + 1)), g));

    ///printf("trial test=%d \ t %.9f =%.9f \ t %.9f \ n ", model, parameters[1], Cr
    cfcall = RCmul(exp(-rf * dt), Cdiv(term1, term2));

    return cfcall;
}

```

```

dcomplex cfCDF(int model, double dt, dcomplex g, double aa, double parameters[])
{
    //double m;
    dcomplex cf, caa, raa, cden;

    caa = Complex(0.0, aa);
    raa = Complex(aa, 0.0);
    cf = cfLevy(model, dt, Csub(g, caa), parameters);

    cden = Cadd(Cmul(Complex(0, 1), g), raa);

    return Cdiv(cf, cden);
}

```

```

dcomplex cfrnshifted(int model, double aa, double rf, double dt, dcomplex g, double parameters[])
{
    dcomplex term1, term2;

```

```

    term1 = cfrn(model, rf, dt, g, parameters);
    term2 = Cexp(Cmul(Complex(0, -aa), g));

    return Cmul(term1, term2);
}

////Moments
double MomentsGauss(int moment, double rf, double dt, double sg)
{
    double mom = 0.;

    if (moment == 1)
    {
        mom = dt * (rf - POW(sg, 2) / 2.);
    }

    if (moment == 2)
    {
        mom = dt * (POW(sg, 2) + dt * POW(rf - POW(sg, 2) / 2., 2));
    }

    return mom;
}

//NIG
double MomentsNig(int moment, double rf, double dt, double alpha, double beta,
{
    double mom = 0., beta2, alpha2;
    double eexp = 2.718281828459045;
    beta2 = beta * beta;
    alpha2 = alpha * alpha;

    if (moment == 1)
    {
        // mom=(dt*(beta*delta + (-alpha2 + beta2 +
        //          sqrt((alpha2 - beta2)*(alpha2 - (1 + beta)*(1+beta))))*delta +
        //  sqrt(alpha2 - beta2)*rf))/sqrt(alpha2 - beta2);

        mom = dt * ((beta * delta) / sqrt(alpha2 - beta2) + rf -

```

```

        log(POW(eexp, -((-sqrt(alpha2 - beta2) + sqrt(alpha2 - POW(1 +
    }

    if (moment == 2)
    {
        mom = (POW(beta, 2) * delta * dt) / POW(POW(alpha, 2) - POW(beta, 2), 1.5)
        POW(dt, 2) * POW((beta * delta) / sqrt(POW(alpha, 2) - POW(beta, 2))
        sqrt(POW(alpha, 2) - POW(1 + beta, 2)) * delta + rf
    }

    return mom;

}

double sec(double x)
{
    return 1 / cos(x);
}

//meixner
double MomentsMeixner(int moment, double rf, double dt, double alpha, double beta)
{
    double mom = 0.;

    if (moment == 1)
    {
        mom = dt * (rf - log(POW(cos(beta / 2.) * sec((alpha + beta) / 2.), 2 * de
    }

    if (moment == 2)
    {
        mom = (dt * (POW(alpha, 2) * delta * POW(sec(beta / 2.), 2) +
        2 * dt * POW(rf - log(POW(cos(beta / 2.) * sec((alpha + beta)
    }

    return mom;

}

double MomentsVarianceGamma(int moment, double rf, double dt, double sg, double
{
    double mom = 0.;

```

```

    if (moment == 1)
    {
        mom = dt * (rf + theta - 1.*log(POW(1. - 0.5 * nu * POW(sg, 2) - 1.*nu * t
    }

    if (moment == 2)
    {
        mom = dt * (1.*POW(sg, 2) + nu * POW(theta, 2) + dt * POW(rf + theta, 2) +
            dt * POW(log(POW(1 - 0.5 * nu * POW(sg, 2) - 1.*nu * theta, -1
            2.*dt * (rf + theta) * log(POW(1. - 0.5 * nu * POW(sg, 2) - 1.
    }

    return mom;

}

//'cgmy
double MomentsCgmy(int moment, double rf, double dt, double ccc, double ggg, do
{
    double mom = 0.;

    if (moment == 1)
    {
        mom = (dt * (ggg * mmm * rf - ccc * (-(POW(ggg, 1 + yyy) * mmm) - POW(ggg,
            ggg * ((POW(1 + ggg, yyy) + POW(-1 +
    }

    if (moment == 2)
    {
        mom = (dt * (ccc * (POW(ggg, yyy) * POW(mmm, 2) + POW(ggg, 2) * POW(mmm, y
            dt * POW(ggg * mmm * rf - ccc * (-(POW(ggg, 1 + yyy) * mmm) -
            ggg * ((POW(1 + ggg, yyy) + POW(-1 + mmm, yyy)) * mm
            exp(pnl_lgamma(-yyy)), 2))) / (POW(ggg, 2) * POW(mmm

    }

    return mom;

}

//'de

```

```

double MomentsDe(int moment, double rf, double dt,
                double sg, double lambda, double p, double eta1, double eta2)
{
    double mom = 0.;

    if (moment == 1)
    {
        mom = (dt * (eta2 * lambda * p + eta1 * (-lambda + lambda * p + eta2 * rf)
                    eta1 * eta2 * ((lambda * (1 + eta1 * (-1 + p) + eta2 * p)) /
    }

    if (moment == 2)
    {
        mom = dt * (-(lambda * ((2 * (-1 + p)) / POW(eta2, 2) - (2 * p) / POW(eta1
                    (dt * POW(eta2 * lambda * p + eta1 * (lambda * (-1 + p) + eta2
                    eta1 * eta2 * ((lambda * (1 + eta1 * (-1 + p) + eta2
    }

    return mom;
}

double MomentsMerton(int moment, double rf, double dt,
                    double sg, double alpha, double lambda, double delta)
{
    double mom = 0.;

    if (moment == 1)
    {
        mom = dt * (alpha * lambda - (-1 + exp(alpha + delta * delta / 2.)) * lamb
    }

    if (moment == 2)
    {
        mom = POW(alpha, 2) * dt * lambda + POW(delta, 2) * dt * lambda + dt * POW
                    (POW(dt, 2) * POW(2 * (-1 - alpha + exp(alpha + delta * delta / 2.))
    }

    return mom;
}

```

```
}
```

```
double MomentsLevy(int model, double rf, int moment, double dt, double parameter  
{
```

```
    double mom = 0.;
```

```
    if (model == 1)
```

```
    {
```

```
        mom = MomentsGauss(moment, rf, dt, parameters[1]);
```

```
    }
```

```
    if (model == 2)
```

```
    {
```

```
        mom = MomentsNig(moment, rf, dt, parameters[ 1], parameters[ 2], parameter
```

```
    }
```

```
    if (model == 3)
```

```
    {
```

```
        mom = MomentsMeixner(moment, rf, dt, parameters[ 1], parameters[ 2], param
```

```
    }
```

```
    if (model == 4)
```

```
    {
```

```
        mom = MomentsVarianceGamma(moment, rf, dt, parameters[ 1], parameters[ 2],
```

```
    }
```

```
    if (model == 5)
```

```
    {
```

```
        mom = MomentsCgmy(moment, rf, dt, parameters[ 1] , parameters[ 2], paramet
```

```
    }
```

```
    if (model == 6)
```

```
    {
```

```
        mom = MomentsDe(moment, rf, dt, parameters[ 1] , parameters[ 2] , paramete
```

```
    }
```

```
    if (model == 7)
```

```

    {
        mom = MomentsMerton(moment, rf, dt, parameters[ 1] , parameters[ 2], param
    }

return mom;
}

dcomplex cfrnstandardized(int model, double rf, double dt, dcomplex g, double pa
{
    double std, m1, m2;

    dcomplex result;

    m1 = MomentsLevy(model, rf, 1, dt, parameters);
    m2 = MomentsLevy(model, rf, 2, dt, parameters);

    std = POW(m2 - m1 * m1, 0.5);

    result = cfrn(model, rf, dt, RCMul(1.0 / std, g), parameters);

    return result;
}

///Pr[L>x]<inf(i) E(L^i/exp(i*x))
double BoundUpperTailLevy(int model, double x, double rf, double dt, int maxmo
{
    double minup, bound;
    int i;

    minup = 1.0;
    for (i = 1; i < maxmoment + 1; i++)
    {
        bound = Creal(cfrn(model, rf, dt, Complex(0, -i), parameters)) / exp(x * i

        minup = MIN(minup, bound);
        // printf("%d b%.12f min%.12f\ n",i, bound,minup);
    }
}

```

```
    return minup;
}

double BoundLowerTailLevy(int model, double x, double rf, double dt, int maxmo
{
    double minlow, bound;
    int i;
    minlow = 1.0;

    for (i = 1; i < maxmoment + 1; i++)
    {
        bound = Creal(cfrn(model, rf, dt, Complex(0, i), parameters)) / exp(x * i)
        minlow = MIN(minlow, bound);
    }

    return minlow;
}

#endif //PremiaCurrentVersion
```