

[Help](#)

```

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
#else

#include <iostream>
#include <vector>

using namespace std;

#include "fft.h"
#include "numerics.h"
#include "levy.h"

double VG_measure::integrated_nu(const double a, const double b) const
{
    if (a >= epsilon)
        return (expint(1, (B - A) * a) - expint(1, (B - A) * b)) / kappa;
    else if (b <= -epsilon)
        return (expint(1, -(A + B) * b) - expint(1, -(A + B) * a)) / kappa;
    else if (a < -epsilon)
    {
        if (b <= epsilon)
            return (expint(1, (A + B) * epsilon) - expint(1, -(A + B) * a)) / kappa;
        else
            return (expint(1, (A + B) * epsilon) - expint(1, -(A + B) * a)
                    + expint(1, (B - A) * epsilon) - expint(1, (B - A) * b)) / kappa;
    }
    else if (b > epsilon)
        return (expint(1, (B - A) * epsilon) - expint(1, (B - A) * b)) / kappa;
    else return 0;
}

VG_measure::VG_measure(const double dtheta, const double dsigma, const double dk
                        const double ddx)
:
theta(dtheta), sigma(dsigma), kappa(dkappa)
{
    drift = log(1 - (theta + sigma * sigma / 2) * kappa) / kappa;

```

```

A = theta / sigma / sigma;
B = sqrt(theta * theta + 2 * sigma * sigma / kappa) / sigma / sigma;
double etap = B - A;
double etam = B + A;
if (etap <= 1) myerror("Error: etap <= 1!");

dx = ddx;
if (dx <= 0) myerror("dx <= 0");

/* approximation of small jumps */
double C = 0.85; // one can choose another constant
epsilon = (ceil(C / sqrt(dx) - 0.5) + 0.5) * dx; // epsilon is of order sqrt(dx)
sigmadiff_squared = pnl_tgamma(2) / kappa * (gammp(2, etam * epsilon) / etam /
      + gammp(2, etap * epsilon) / etap / etap);

/* truncation of large jumps */
const double tolerance = 0.00001;
Kmin = (int) floor(-log(1. / tolerance) / etam / dx);
Kmax = (int) ceil(log(1. / tolerance) / etap / dx);
double ymin = (Kmin - 0.5) * dx;
if (ymin >= -epsilon)
{
    Kmin = (int)(-epsilon / dx - 0.5);
    ymin = (Kmin - 0.5) * dx;
}
double ymax = (Kmax + 0.5) * dx;
if (ymax <= epsilon)
{
    Kmax = (int)(epsilon / dx + 0.5);
    ymax = (Kmax + 0.5) * dx;
}

espX1 = theta + log(1 - sigma * sigma * kappa / 2 - theta * kappa) / kappa;
varX1 = sigma * sigma + theta * theta * kappa;

nu_array = new std::vector<double> (Kmax - Kmin + 1); //nu_array[j-Kmin] = nu(j)

for (int j = Kmin; j <= Kmax; j++)
{
    (*nu_array)[j - Kmin] = integrated_nu((j - 0.5) * dx, (j + 0.5) * dx);
}

```

```

    }

    lambda = integrated_nu(ymin, -epsilon) + integrated_nu(epsilon, ymax);
    alpha = (expint(1, (etap - 1) * epsilon) - expint(1, (etap - 1) * ymax)
             + expint(1, (etam + 1) * epsilon) - expint(1, -(etam + 1) * ymin)) /
}

VG_measure::~VG_measure()
{
    delete nu_array;
}

/*-----

double NIG_measure::integrated_nu(const double a, const double b) const
/* uses routine of numerical integration qromb from numerics.h;
if dx is very small, the quadrature error may become dominant:
in this case, try to diminish the constant EPS in qromb */
{
    if (a >= b) myerror("in integrated_nu a>=b");
    if ((a >= epsilon) || (b <= -epsilon))
        return qromb(Ref_Levy_measure(*this), a, b);
    else if (a < -epsilon)
    {
        if (b <= epsilon)
            return qromb(Ref_Levy_measure(*this), a, -epsilon);
        else
        {
            return qromb(Ref_Levy_measure(*this), a, -epsilon)
                + qromb(Ref_Levy_measure(*this), epsilon, b);
        }
    }
    else if (b > epsilon)
        return qromb(Ref_Levy_measure(*this), epsilon, b);
    else return 0;
}

NIG_measure::NIG_measure(const double dtheta, const double dsigma, const double
                        const double ddx)
:

```

```

    theta(dtheta), sigma(dsigma), kappa(dkappa)
{
    drift = (sqrt(1 - sigma * sigma * kappa - 2 * theta * kappa) - 1) / kappa;

    A = theta / sigma / sigma;
    B = sqrt(theta * theta + sigma * sigma / kappa) / sigma / sigma;
    C = sqrt(theta * theta + sigma * sigma / kappa) / (M_PI * sigma * sqrt(kappa))
    /* auxiliary parameters describing the decrease of tails of nu at infinity */
    double etap = B - A; // eta_+
    double etam = B + A; // eta_-

    dx = ddx;

    /* approximation of small jumps */
    double CC = 1; // one can chose another constant
    int keps = (int)ceil(CC / sqrt(dx) + 0.5);
    epsilon = (keps - 0.5) * dx; // epsilon is of order sqrt{dx}

    sigmadiff_squared = qromb(NIG_nu_x2(*this), -epsilon, epsilon);

    /* truncation of large jumps */
    const double tolerance = 0.00001;
    Kmin = (int) floor(-log(1. / tolerance) / etam / dx);
    Kmax = (int) ceil(log(1. / tolerance) / etap / dx);
    double ymin = (Kmin - 0.5) * dx;
    if (ymin >= -epsilon)
    {
        Kmin = (int)(-epsilon / dx - 0.5);
        ymin = (Kmin - 0.5) * dx;
    }
    double ymax = (Kmax + 0.5) * dx;
    if (ymax <= epsilon)
    {
        Kmax = (int)(epsilon / dx + 0.5);
        ymax = (Kmax + 0.5) * dx;
    }

    espX1 = theta + drift;
    varX1 = sigma * sigma + theta * theta * kappa;

```

```

nu_array = new std::vector<double> (Kmax - Kmin + 1); //nu_array[j-Kmin] = nu(

lambda = 0;
alpha = 0;
for (int j = Kmin; j <= -keps; j++)
{
    double xjm = (j - 0.5) * dx;
    double xjp = (j + 0.5) * dx;
    (*nu_array)[j - Kmin] = integrated_nu(xjm, xjp);
    lambda += (*nu_array)[j - Kmin];
    alpha += qromb(Levy_nu_expx(*this), xjm, xjp);
}
for (int j = keps; j <= Kmax; j++)
{
    double xjm = (j - 0.5) * dx;
    double xjp = (j + 0.5) * dx;
    (*nu_array)[j - Kmin] = integrated_nu(xjm, xjp);
    lambda += (*nu_array)[j - Kmin];
    alpha += qromb(Levy_nu_expx(*this), xjm, xjp);
}
alpha -= lambda;
}

NIG_measure::~NIG_measure()
{
    delete nu_array;
}

/*-----

double TS_measure::integrated_nu(const double a, const double b) const
/* uses routine of numerical integration qromb from numerics.h;
if dx is very small, the quadrature error may become dominant:
in this case, try to diminish the constant EPS in qromb */
{
    if (a >= b) myerror("in integrated_nu a>=b");
    if (a >= epsilonp)
        if (alphap == 1)
            return cp * (exp(-lambdap * a) / a - exp(-lambdap * b) / b
                        + lambdap * (expint(1, lambdap * b) - expint(1, lambdap * a)
            else if (alphap == 0)

```

```

        return cp * (expint(1, a * lambdap) - expint(1, b * lambdap));
    else
        return cp * ((exp(-lambdap * a) / pow(a, alphap) * (1 - lambdap * a / (alp
            exp(-lambdap * b) / pow(b, alphap) * (1 - lambdap * b / (alp
            + pow(lambdap, alphap) / alphap / (alphap - 1) * pnl_tgamma(2
            * (gammq(2 - alphap, lambdap * a) - gammq(2 - alphap, lambdap
    else if (b <= -epsilon_m)
        if (alpham == 1)
            return cm * (-exp(lambdam * b) / b + exp(lambdam * a) / a
                + lambdam * (expint(1, -lambdam * a) - expint(1, -lambdam *
        else if (alpham == 0)
            return cm * (expint(1, -b * lambdam) - expint(1, -a * lambdam));
        else
            {
                return cm * ((exp(lambdam * b) / pow(-b, alpham) * (1 + lambdam * b / (a
                    exp(lambdam * a) / pow(-a, alpham) * (1 + lambdam * a / (a
                    + pow(lambdam, alpham) / alpham / (alpham - 1) * pnl_tgamma
                    * (gammq(2 - alpham, -lambdam * b) - gammq(2 - alpham, -lam
            }
    else if (a < -epsilon_m)
    {
        if (b <= epsilon_p)
            if (alpham == 1)
                return cm * (exp(-lambdam * epsilon_m) / epsilon_m + exp(lambdam * a) /
                    + lambdam * (expint(1, -lambdam * a) - expint(1, lambdam
            else if (alpham == 0)
                return cm * (expint(1, epsilon_m * lambdam) - expint(1, -a * lambdam));
            else
                return cm * ((exp(-lambdam * epsilon_m) / pow(epsilon_m, alpham) * (1 -
                    exp(lambdam * a) / pow(-a, alpham) * (1 + lambdam * a /
                    + pow(lambdam, alpham) / alpham / (alpham - 1) * pnl_tgamma
                    * (gammq(2 - alpham, lambdam * epsilon_m) - gammq(2 - alph
        else
        {
            double ans = 0;
            if (alpham == 1)
                ans += cm * (-exp(lambdam * epsilon_m) / epsilon_m + exp(lambdam * a)
                    + lambdam * (expint(1, -lambdam * a) - expint(1, lambda
            else if (alpham == 0)
                ans += cm * (expint(1, epsilon_m * lambdam) - expint(1, -a * lambdam)
            else

```

```

        ans += cm * ((exp(-lambdam * epsilon_m) / pow(epsilon_m, alphas) * (1
            exp(lambdam * a) / pow(-a, alphas) * (1 + lambdam * a
            + pow(lambdam, alphas) / alphas / (alphas - 1) * pnt_tg
            * (gammq(2 - alphas, lambdam * epsilon_m) - gammq(2 - al
if (alphap == 1)
    ans += cp * (exp(-lambdap * epsilon_p) / epsilon_p - exp(-lambdap * b)
        + lambdap * (expint(1, lambdap * b) - expint(1, lambdap
else if (alphap == 0)
    ans += cp * (expint(1, epsilon_p * lambdap) - expint(1, b * lambdap))
else
    ans += cp * ((exp(-lambdap * epsilon_p) / pow(epsilon_p, alphap) * (1
        exp(-lambdap * b) / pow(b, alphap) * (1 - lambdap * b
        + pow(lambdap, alphap) / alphap / (alphap - 1) * pnt_tgamma(2
        * (gammq(2 - alphap, lambdap * epsilon_p) - gammq(2 - alphap,
    return ans;
}
}
else if (b > epsilon_p)
    if (alphap == 1)
        return cp * (exp(-lambdap * epsilon_p) / epsilon_p - exp(-lambdap * b) / b
            + lambdap * (expint(1, lambdap * b) - expint(1, lambdap * eps
    else if (alphap == 0)
        return cp * (expint(1, epsilon_p * lambdap) - expint(1, b * lambdap));
    else
        return cp * ((exp(-lambdap * epsilon_p) / pow(epsilon_p, alphap) * (1 - lamb
            exp(-lambdap * b) / pow(b, alphap) * (1 - lambdap * b / (alp
            + pow(lambdap, alphap) / alphap / (alphap - 1) * pnt_tgamma(2
            * (gammq(2 - alphap, lambdap * epsilon_p) - gammq(2 - alphap,
    else return 0;

}

TS_measure::TS_measure(const double dalphas, const double dalphas,
                        const double dlambdap, const double dlambdam,
                        const double dcp, const double dcm, const double ddx)
:
    alphas(dalphas), alphas(dalphas),
    lambdap(dlambdap), lambdam(dlambdam), cp(dcp), cm(dcm)
{
    if ((alphap <= 0) || (alphap >= 2)) myerror("invalid parameter alphap");
    if ((alpham <= 0) || (alpham >= 2)) myerror("invalid parameter alpham");

```

```

if (lambdap <= 1) myerror("lambdap <= 1!");
if (lambdam <= 0) myerror("lambdam <= 0!");
if (cp <= 0) myerror("cp <= 0!");
if (cm <= 0) myerror("cm <= 0!");

drift = 0;
if (alphap == 1)
    drift -= cp * (lambdap - 1.) * log(1. - 1. / lambdap);
else
    drift -= pnl_tgamma(-alphap) * pow(lambdap, alphap) * cp
            * (pow(1. - 1. / lambdap, alphap) - 1. + alphap / lambdap);
if (alphan == 1)
    drift -= cm * (lambdam + 1.) * log(1. + 1. / lambdam);
else
    drift -= pnl_tgamma(-alphan) * pow(lambdam, alphan) * cm
            * (pow(1. + 1. / lambdam, alphan) - 1. - alphan / lambdam);

dx = ddx;
if (dx <= 0) myerror("dx <= 0");

/* approximation of small jumps */
double C = 1; //one can chose another constant
double ap = (alphap <= 1) ? 1. / (3 - alphap) : 1. / (1 + alphap);
double am = (alphan <= 1) ? 1. / (3 - alphan) : 1. / (1 + alphan);
epsilonp = (ceil(C * pow(dx, ap - 1) + 0.5) - 0.5) * dx;
epsilonm = (ceil(C * pow(dx, am - 1) + 0.5) - 0.5) * dx;
sigmadiff_squared = cp / pow(lambdap, 2 - alphap) * pnl_tgamma(2 - alphap) * g
                    + cm / pow(lambdam, 2 - alphan) * pnl_tgamma(2 - alphan) * g

/* truncation of large jumps */
const double tolerance = 0.00001;
Kmin = (int) floor(-log(1. / tolerance) / lambdam / dx);
Kmax = (int) ceil(log(1. / tolerance) / lambdap / dx);
double ymin = (Kmin - 0.5) * dx;
if (ymin >= -epsilonm)
{
    Kmin = (int)(-epsilonm / dx - 0.5);
    ymin = (Kmin - 0.5) * dx;
}
double ymax = (Kmax + 0.5) * dx;
if (ymax <= epsilonp)

```



```

{
    Kmax = (int)(epsilonp / dx + 0.5);
    ymax = (Kmax + 0.5) * dx;
}

espX1 = 0;
if (alpham != 1) espX1 -= pnl_tgamma(-alpham) * pow(lambdam, alpham) * cm
                        * (pow(1 + 1. / lambdam, alpham) - 1 - alpham / la
else espX1 -= cm * ((lambdam + 1) * log(1 + 1. / lambdam) - 1);

if (alphap != 1) espX1 -= pnl_tgamma(-alphap) * pow(lambdap, alphap) * cp
                        * (pow(1 - 1. / lambdap, alphap) - 1 + alphap / la
else espX1 -= cp * ((lambdap - 1) * log(1 - 1. / lambdap) + 1);

varX1 = pnl_tgamma(2 - alphap) * cp / pow(lambdap, 2 - alphap) + pnl_tgamma(2
nu_array = new std::vector<double> (Kmax - Kmin + 1); //nu_array[j-Kmin] = nu(

for (int j = Kmin; j <= Kmax; j++)
{
    (*nu_array)[j - Kmin] = integrated_nu((j - 0.5) * dx, (j + 0.5) * dx);
}
lambda = integrated_nu(ymin, -epsilonm) + integrated_nu(epsilonp, ymax);

alpha = 0;
if (alphap == 1)
    alpha += cp * (exp(-(lambdap - 1) * epsilonp) / epsilonp - exp(-(lambdap -
                        + (lambdap - 1) * (expint(1, (lambdap - 1) * ymax) - expint(
else if (alphap == 0)
    alpha += cp * expint(1, epsilonp * (lambdap - 1));
else
    alpha += cp * ((exp(-(lambdap - 1) * epsilonp) / pow(epsilonp, alphap) * (1
                        exp(-(lambdap - 1) * ymax) / pow(ymax, alphap) * (1 - (lambd
                        + pow(lambdap - 1, alphap) / alphap / (alphap - 1) * pnl_tgam
                        * (gammq(2 - alphap, (lambdap - 1) * epsilonp) - gammq(2 - al

if (alpham == 0)
    alpha += cm * expint(1, epsilonm * (1 + lambdam));
else if (alpham == 1)
    alpha += cm * (exp(-(lambdam + 1) * epsilonm) / epsilonm + exp((lambdam + 1)

```

```

        + (lambdam + 1) * (expint(1, -(lambdam + 1) * ymin) - expint(
else
{
    alpha += cm * ((exp(-(lambdam + 1) * epsilon) / pow(epsilon, alpham) * (
        * epsilon / (alpham - 1)) -
        exp((lambdam + 1) * ymin) / pow(-ymin, alpham) * (1 + (lam
    + pow(lambdam + 1, alpham) / alpham / (alpham - 1) * pnl_tg
    * (gamma(2 - alpham, (lambdam + 1) * epsilon) - gamma(2 -
}

    alpha -= lambda;
}

TS_measure::~TS_measure()
{
    delete nu_array;
}

complex<double> TS_measure::cf(const double T, const complex<double> &u) const
{
    complex<double> result = 0;

    if (alphap == 1)
        result += cp * (lambdap - I * u) * log(1. - I * u / lambdap);
    else
        result += pnl_tgamma(-alphap) * pow(lambdap, alphap) * cp
            * (pow(1. - I * u / lambdap, alphap) - 1. + I * u * alphap / lambd
    if (alpham == 1)
        result += cm * (lambdam + I * u) * log(1. + I * u / lambdam);
    else
        result += pnl_tgamma(-alpham) * pow(lambdam, alpham) * cm
            * (pow(1. + I * u / lambdam, alpham) - 1. - I * u * alpham / lambd

    return exp(T * (result + I * u * drift));
}

/*-----

Merton_measure::Merton_measure(const double dm, const double ddelta, const double
    const double sigma, const double ddx)

```

```

:
mu(dmu), delta(ddelta), factor(dfactor)
{
    sigmadiff_squared = sigma * sigma;
    drift = -sigmadiff_squared / 2 - factor * (exp(mu + delta * delta / 2) - 1);
    dx = ddx;
    if (dx <= 0) myerror("dx <= 0");

    /* truncation of large jumps */
    const double Nsupp = 6; //we limit the support of nu to (mu-Nsupp*pnl_tgamma,m
    Kmin = (int) floor((mu - Nsupp * delta) / dx);
    Kmax = (int) ceil((mu + Nsupp * delta) / dx);

    espX1 = - sigmadiff_squared / 2 - factor * (exp(mu + delta * delta / 2) - 1 -
    varX1 = sigmadiff_squared + factor * (delta * delta + mu * mu);

    nu_array = new std::vector<double> (Kmax - Kmin + 1); //nu_array[j-Kmin] = nu(

    for (int j = Kmin; j <= Kmax; j++)
    {
        (*nu_array)[j - Kmin] = integrated_nu((j - 0.5) * dx, (j + 0.5) * dx);
    }
    double ymin = (Kmin - 0.5) * dx;
    double ymax = (Kmax + 0.5) * dx;
    lambda = integrated_nu(ymin, ymax);
    alpha = factor * exp(mu + delta * delta / 2) * (normCDF((ymax - mu - delta * d
        normCDF((ymin - mu - delta * delta) / delta)) - lambda;

}

Merton_measure::~Merton_measure()
{
    delete nu_array;
}

/*-----
Kou_measure::Kou_measure(const double dfactor, const double dlambdap,
                        const double dlambdam, const double dp,
                        const double sigma, const double ddx)
:

```

```

    factor(dfactor), lambdap(dlambdap), lambdam(dlambdam), p(dp)
{
    if (lambdap <= 1) myerror("lambdap <= 1!");
    sigmadiff_squared = sigma * sigma;
    drift = -sigmadiff_squared / 2 - factor * (p / (lambdap - 1) - (1 - p) / (lambdam - 1));
    dx = ddx;
    if (dx <= 0) myerror("dx <= 0");

    /* truncation of large jumps */
    const double tolerance = 0.00001;
    Kmin = (int) floor(-log(1. / tolerance) / lambdam / dx);
    Kmax = (int) ceil(log(1. / tolerance) / lambdap / dx);

    espX1 = - sigmadiff_squared / 2 - factor * (p / lambdap / (lambdap - 1) + (1 - p) / (lambdam - 1));
    varX1 = sigmadiff_squared + factor * (p / lambdap / lambdap + (1 - p) / lambdam);

    nu_array = new std::vector<double> (Kmax - Kmin + 1); //nu_array[j-Kmin] = nu(j);

    for (int j = Kmin; j <= Kmax; j++)
    {
        (*nu_array)[j - Kmin] = integrated_nu(j * dx - dx / 2, j * dx + dx / 2);
    }

    double ymin = (Kmin - 0.5) * dx;
    double ymax = (Kmax + 0.5) * dx;
    lambda = integrated_nu(ymin, ymax);
    alpha = p * factor * lambdap * (1 - exp(-(lambdap - 1) * ymax)) / (lambdap - 1)
            + (1 - p) * factor * lambdam * (1 - exp(-(1 + lambdam) * ymin)) / (1 + lambdam);

}

Kou_measure::~Kou_measure()
{
    delete nu_array;
}

#endif //PremiaCurrentVersion

```