

### Help

```

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
    (2007+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else

#ifdef LEVY_H
#define LEVY_H

#include <iostream>
#include <fstream>
#include <complex>
#include <vector>
#include "progonka.h"
#include "numerics.h"

const complex<double> I(0,1);

class Levy_measure
/* generic class for a Levy model */
{
    virtual double nu(const double x) const = 0; // value of
        the Levy density nu at x
    virtual double integrated_nu(const double a, const
        double b) const = 0; // integral of nu(dx) on the interval (a,b)

protected:
    double drift; // used in calculation of the characteristi
        c function of X_T

public:
    double lambda; // nu(R)
    double alpha; // integral of (exp(x)-1) with respect to
        nu(dx) on R
    double sigmadiff_squared; // variance of the diffusion
        part
    double espX1; // expectation of X1
    double varX1; // variance of X1
    vector<double>* nu_array; // values of nu on the grid:
        nu_array[j-Kmin] = nu(xj)*dx, j=Kmin,...,Kmax.
    int Kmin;

```

```

int Kmax;
double dx; // space discretization step

virtual complex<double> cf(const double T, const complex<
    double> & u) const = 0; // characteristic function of  $X_T$ 

friend class Ref_Levy_measure; /* an auxiliary class wh
    ich allows to substitute the function-member double nu(
    double x)
        as an argument of another function (we us
        e it in the function-template
                                double qromb(T func,
    double a, double b) from numerics.h) */
friend class Levy_nu_expx; /* an auxiliary class which
    allows to use the function-template
        double qromb(T func, double a, double b) from numerics.h
        with func(x) = exp(x)*nu(x) */
virtual ~Levy_measure(){};
};

class Ref_Levy_measure
{
    const Levy_measure & refmeasure;

public:
    Ref_Levy_measure(const Levy_measure & measure) : refmeasure(
        measure) {}

    double operator() (const double x) const {return refmea
        sure.nu(x);}
};

class Levy_nu_expx
{
    const Levy_measure & refmeasure;

public:
    Levy_nu_expx(const Levy_measure & measure) : refmeasure(
        measure) {}

    double operator() (const double x) const {return exp(x)

```

```

        *refmeasure.nu(x);}
};

/* We derive below classes for specific Levy models from the
   general class Levy_measure */

class Merton_measure : public Levy_measure
{
    double mu;
    double delta;
    double factor;

    double nu(const double x) const
    {
        return factor*normPDF((x-mu)/delta)/delta;
    }

    double integrated_nu(const double a, const double b) const
    {
        return factor*(normCDF((b-mu)/delta)-normCDF((a-mu)/delta));
    }

public:

    Merton_measure(const double dmu, const double ddelta,
        const double dfactor, const double sigma,
        const double ddx);
    virtual ~Merton_measure();

    complex<double> cf(const double T, const complex<double>
        & u) const
    {
        return exp(T*(-sigmadiff_squared*u*u/2.+factor*(exp(
            I*mu*u-delta*delta*u*u/2.))-1.)
            +I*u*drift));
    }
};

/*-----

```

```

-----*/
class Kou_measure : public Levy_measure
{
    double factor;
    double lambdap;
    double lambdam;
    double p;

    double nu(const double x) const
    {
        return (x>0) ? p*factor*lambdap*exp(-lambdap*x)
            : (1-p)*factor*lambdam*exp(lambdam*x);
    }

    double integrated_nu(const double a, const double b) const
    {
        if (a>=b) myerror("in function integrated_nu a must
            be less than b");
        else if (a >= 0)
            return p*factor*(exp(-lambdap*a)-exp(-lambdap*b));
        else if (b <= 0)
            return (1-p)*factor*(exp(lambdam*b)-exp(lambdam*a));
        else
            return factor*(1-p*exp(-lambdap*b)-(1-p)*exp(lambdam*
                a));

        /* to avoid a warning */
        return 0;
    }
}

public:

    Kou_measure(const double dfactor, const double dlambdap,
        const double dlambdam, const double dp,
        const double sigma, const double ddx);
    virtual ~Kou_measure();
    complex<double> cf(const double T, const complex<double>
        & u) const
    {
        return exp(T*(-sigmadiff_squared*u*u/2. + I*u*
            factor*p/(lambdap-I*u)

```

```

        - I*u*factor*(1-p)/(lambdam+I*u)+I*u*drift));
    }
};

/*-----
-----*/
class VG_measure : public Levy_measure
{
    double theta;
    double sigma;
    double kappa;
    double A;
    double B;
    double epsilon;

    double nu(const double x) const
    {
        if (fabs(x)>=epsilon){
            return x>0 ? exp((A-B)*x)/kappa/x : -exp((A+B)*x)/kappa/x;
        }
        else return 0;
    }
    double integrated_nu(const double a, const double b) const;

public:
    VG_measure(const double dtheta, const double dsigma,
               const double dkappa, const double ddx);
    virtual ~VG_measure();
    complex<double> cf(const double T, const complex<double>
                      & u) const
    {
        return exp(-T/kappa*log(1.-I*theta*kappa*u+sigma*sigma*kappa/2*u*u)
                  + I*T*u*drift);
    }
};

/*-----
-----*/

```

```

-----*/
class NIG_measure : public Levy_measure
{
    double theta;
    double sigma;
    double kappa;
    double A;
    double B;
    double C;
    double epsilon;
    double nu(const double x) const
    {
        if (fabs(x)>=epsilon){
            return (A*x < 600) ? C/fabs(x)*exp(A*x)*bessk1(B*fabs
            s(x)) // to avoid overflow due to exp(Ax)
            : C*sqrt(M_PI/(2*B*abs(x)))*exp(A*x-B*abs(x))/abs(x);
        }
        else return 0;
    }
    double integrated_nu(const double a, const double b) cons
    t;

public:

    NIG_measure(const double dtheta, const double dsigma,
        const double dkappa, const double ddx);
    virtual ~NIG_measure();

    complex<double> cf(const double T, const complex<double>
    & u) const
    {
        return exp(T*(1.-sqrt(1.+u*u*sigma*sigma*kappa-2.*I*
        theta*u*kappa))/kappa
        + I*T*u*drift);
    }

    friend class NIG_nu_x2; /* an auxiliary class which all
    ows to use the function-template
        double qromb(T func, double a, double b) fro
    m numerics.h
        with func(x) = x^2*nu(x) */

```

```

};

class NIG_nu_x2
{
    const NIG_measure & refmeasure;

public:
    NIG_nu_x2(const NIG_measure & measure) : refmeasure(measu
        re) {}

    double operator() (const double x) const {
        if (refmeasure.A*x < 600) // to avoid overflow due
            to exp(Ax)
        return x==0 ? refmeasure.C/refmeasure.B : refmeasure.C*
            abs(x)*exp(refmeasure.A*x)*bessk1(refmeasure.B*abs(x));
        else
        return refmeasure.C*sqrt(M_PI*abs(x)/(2*refmeasure.B))*
            exp(refmeasure.A*x-refmeasure.B*abs(x));
        }
};

/*-----
-----*/
class TS_measure : public Levy_measure
{
    double alphap, alpham;
    double lambdap, lambdam;
    double cp, cm;
    double epsilonp;
    double epsilonm;

    inline double nu(const double x) const
    {
        if (x>=epsilonp) return cp*exp(-lambdap*x)/pow(x,1+
            alphap);
        else if (x<= -epsilonm) return cm*exp(lambdam*x)/pow(
            -x,1+alpham);
        else return 0;
    }
}

```

```
double integrated_nu(const double a, const double b) const;

public:

    TS_measure(const double dalphap, const double dalpham,
               const double dlambdap, const double dlambdam,
               const double dcp, const double dcm, const double dd
               x);
    virtual ~TS_measure();

    complex<double> cf(const double T, const complex<double>
                      & u) const;
};

#endif

#endif //PremiaCurrentVersion
```

## References