

[Help](#)

```

#ifndef __levy_diffusion__
#define __levy_diffusion__

#include "pnl/pnl_vector.h"
#include "pnl/pnl_band_matrix.h"
#include "pnl/pnl_tridiag_matrix.h"

extern dcomplex Ctgamma_log(dcomplex z);

typedef struct _Heston_diffusion Heston_diffusion;

struct _Heston_diffusion
{
    double Eta;
    double Kappa;
    double Rho;
    double Theta;
    double Sigma;
    double sigma_sqr;
    double theta_sqr;
    double sigma_sqr_d_eta_kappa;
    double etakappathetam2;
    double rho_theta;
    double Drift;
    int nb_parameters;
};

extern Heston_diffusion *Heston_diffusion_create(double Eta_,
    double Kappa_, double Rho_,
    double Theta_, double Sigma_,
    double *jump_drift);

extern void Heston_diffusion_list(const Heston_diffusion *
    process);
extern dcomplex Heston_diffusion_characteristic_exponent(dcomplex u,
    double t, void *mod);
extern dcomplex Heston_diffusion_ln_characteristic_function(dcomplex u,
    double t, void *mod);

/*

```

$$\begin{aligned}
dS_t &= (r - q - \{\lambda_y \mu\} S_t dt + \{\sqrt{V_t}\} S_t dW_t^1 \\
&\quad + J_y S_t dq_y(t) \\
dV_t &= \{\kappa_{\nu}\} \{\text{left}(\{\eta_{\nu}\} + V_t \text{right}) + \{\theta_{\eta}\} \\
&\quad a_{\nu}\} \{\sqrt{V_t}\} dW_t^2 \\
dW^1 dW^2 &= \{\rho\} dt
\end{aligned}$$

$(1+J_y)$ is a lognormally distributed with mean $\{\mu_y\}$ and variance $\{\sigma_y^2\}$

$\{q_y\}$ is an independent Poisson process with arrival rate $\{\lambda_y\}$
 $\{\mu = \text{left}(\{\exp\{\mu_y + \{\sigma_y^2/2\} - 1\right)\}.\}$

*/

```
typedef struct _Bates_diffusion Bates_diffusion;
```

```

struct _Bates_diffusion
{
    double Eta;
    double Kappa;
    double Theta;
    double Rho;
    double Sigma;

    double mu_J;
    double Sigma_J;
    double Lambda_J;

    double sigma_sqr;
    double theta_sqr;
    double sigma_sqr_d_eta_kappa;
    double etakappathetam2;
    double rho_theta;
    double lnonepmuj;
    double sigmaj_sqr_demi;
    double Drift;

```

```

int nb_parameters;
};

extern Bates_diffusion *Bates_diffusion_create(double Eta_,
double Kappa_, double Rho_,
double Theta_, double Sigma_,
double mu_J_,
double Sigma_J_, double Lambda_J_, double *jump_drift);
extern dcomplex Bates_diffusion_characteristic_exponent(dcomplex u, double t, void *mod);
extern dcomplex Bates_diffusion_ln_characteristic_function(dcomplex u, double t, void *mod);
/*
@article {MR1841412,
  AUTHOR = {Barndorff-Nielsen, Ole E. and Shephard, Neil},
  TITLE = {Non-Gaussian Ornstein-Uhlenbeck-based models and some of their uses in financial economics},
  JOURNAL = {J. R. Stat. Soc. Ser. B Stat. Methodol.},
  FJOURNAL = {Journal of the Royal Statistical Society. Series B. Statistical Methodology},
  VOLUME = {63},
  YEAR = {2001},
  NUMBER = {2},
  PAGES = {167--241},
  ISSN = {1369-7412},
  MRCLASS = {62M07 (62M09 62M10 62P20)},
  MRNUMBER = {MR1841412 (2002c:62127)},
}

```

The square volatility follows the SDE of the form :

$$d\sigma_t^2 = -\lambda \sigma_t^2 dt + dZ_{\{\lambda t\}}$$

where $\lambda > 0$ and Z is a subordinator.

The risk neutral *dynamic* of the log price $x_t = \log S_t$ are given by

$$dW_t = (r - q - \lambda k(-\rho) - \sigma_t^2/2) dt + \sigma_t dW_t + \rho dZ_t,$$

{quad $x_0 = \log(S_0)$.}

where $k(u) = \{\log\{\{\text{mean}\{\{\exp\{-u z_1\}\}\}\}\}.$
 Choice z_t as a compound poisson process,
 $z_t = \sum_{n=1}^{N_t} x_n$
 where N_t is a Poisson process with intensity parameter α
 and each x_n follows an exponential law with mean $\frac{1}{\beta}$.
 One can show that the process σ_t^2 is a stationary process with a
 marginal law that follows a Gamma distribution with mean α and
 variance $\frac{\alpha}{\beta}$. In this case,
 $k(u) = \frac{-\alpha}{\beta + u}$.

*/

```

typedef struct _BNS_diffusion BNS_diffusion;

struct _BNS_diffusion
{
    double Lambda;
    double Rho;
    double Beta;
    double Alpha;
    double Sigma0;

    double Sigma0_sqr ;
    double Lambda_m1;
    double Drift; // proportional to Drift correction
    int nb_parameters;
};

extern BNS_diffusion *BNS_diffusion_create(double Lambda_,
    double Rho_,
    double Beta_, double Alpha_,
    double Sigma0_, double *jump_drift);
extern dcomplex BNS_diffusion_characteristic_exponent(dcomplex u, double t, void *mod);
extern dcomplex BNS_diffusion_ln_characteristic_function(dcomplex u, double t, void *mod);
  
```

```
extern void BNS_diffusion_list(const BNS_diffusion *proces
    s);
```

```
/*
  @article {MR1793362,
    AUTHOR = {Duffie, Darrell and Pan, Jun and Singleton,
      Kenneth},
    TITLE = {Transform analysis and asset pricing for affine
      jump-diffusions},
    JOURNAL = {Econometrica},
    FJOURNAL = {Econometrica. Journal of the Econometric Soc
      iety},
    VOLUME = {68},
    YEAR = {2000},
    NUMBER = {6},
    PAGES = {1343--1376},
    ISSN = {0012-9682},
    CODEN = {ECMTA7},
    MRCLASS = {91B28 (60J60)},
    MRNUMBER = {MR1793362 (2001m:91081)},
  }
```

```
dS_t = (r-q-{\lambda_y \mu}) S_t dt + {\sqrt{V_t}} S_t dW_t^1 +
  J_y S_t dq_y(t) {\{
dV_t = {\kappa_{\nu}} {\left( {\eta_{\nu}} + V_t {\right)} + {\theta_{
  {\nu}} {\sqrt{V_t}}
dW_t^2 + J_V dq_{{\nu}}(t)
```

```
dW^1 dW^2 = {\rho} dt
```

$(1+J_y)$ is a lognormally distributed with mean μ_y and variance σ_y^2

J_V has an exponential distribution with mean μ_{ν}

$q_{\{y\}}$ and $q_{\{\nu\}}$ are indenpendent Poisson process with arrivals rates

$\lambda_{\{y\}}$ and $\lambda_{\{\nu\}}$

$\mu = {\left({\exp {\mu_y + {\sigma_y^2 / 2}} - 1} \right).}$

```
*/

typedef struct _DPS_diffusion DPS_diffusion;

struct _DPS_diffusion
{
    double Eta;
    double Kappa;
    double Rho;
    double Theta;
    double Sigma;

    double mu_y;
    double Sigma_y_sqr_demi;
    double Lambda_y;

    double mu_v;
    double Lambda_v;

    double sigma_cy_sqr_demi;
    double mu_cy;
    double mu_cv;
    double Lambda_c;
    double rho_j;

    double s_lambda;

    double sigma_sqr;
    double theta_sqr;
    double sigma_sqr_d_eta_kappa;
    double etakappathetam2;
    double rho_theta;

    double Drift;
    int nb_parameters;
};
```

```

extern DPS_diffusion *DPS_diffusion_create(double Eta_,
    double Kappa_,
    double Rho_, double Theta_,
    double Sigma_, double mu_y_,
    double Sigma_y_, double Lambda_y_,
    double mu_v_, double Lambda_v_,
    double mu_cy_, double Sigma_cy_,
    double mu_cv_, double Lambda_c_,
    double rho_j_, double *jump_drift);
extern dcomplex DPS_diffusion_characteristic_exponent(dcomplex u, double t, void *mod);
extern dcomplex DPS_diffusion_ln_characteristic_function(dcomplex u, double t, void *mod);
extern void DPS_diffusion_list(const DPS_diffusion *model);

```

```

/*

```

The two following class of model come from :

```

@article {MR1995283,
  AUTHOR = {Carr, Peter and Geman, H{{'e}}lyette and Madan, Dilip B. and Yor, Marc},
  TITLE = {Stochastic volatility for {L}{'evy processes}},
  JOURNAL = {Math. Finance},
  FJOURNAL = {Mathematical Finance. An International Journal of Mathematics, Statistics and Financial Economics},
  VOLUME = {13},
  YEAR = {2003},
  NUMBER = {3},
  PAGES = {345--382},
  ISSN = {0960-1627},
  MRCLASS = {91B28 (60G51)},
  MRNUMBER = {MR1995283 (2005a:91054)},
}

```

CIR stochastic clock

the CIR process can be use as rate of time change. it follows the SDE

$$dy_t = \{\kappa \left(\eta - y_t \right) dt + \{\lambda \sqrt{y_t} dW_t\}$$

```

*/
typedef struct _CIR_diffusion CIR_diffusion;

struct _CIR_diffusion
{
    double Kappa;
    double Eta;
    double Lambda;
    double y0;
    double Drift; // proportional to Drift correction
    double Kappa_sqr;
    double Lambda_sqr;
    double Kappa_sqr_eta_div_lambda_sqr;
    double Two_kappa_eta_div_lambda_sqr;

    double time;
    double Jump_drift;
    double Jump_drift_psi;
    int nb_parameters;
    void *Levy;
    dcomplex(*characteristic_exponent)(dcomplex u, void *mod)
        ;
};

extern CIR_diffusion *CIR_diffusion_create(double Kappa,
    double Eta, double Lambda, double y0,
    void *Levy_,
    dcomplex(*characteristic_exponent_)(dcomplex, void *),
    double *jump_drift);

extern void CIR_diffusion_list(const CIR_diffusion *processes);

extern dcomplex CIR_diffusion_characteristic_exponent(dcomplex u, double t, void *mod);

```



```

extern dcomplex CIR_diffusion_ln_characteristic_function(dcomplex u, double t, void *mod);
extern double CIR_diffusion_get_sigma_square(CIR_diffusion *Process);
extern void CIR_diffusion_fourier_stiffness(CIR_diffusion *mod, double hx, double bnd_fourier, int Nw, int kmin, int kmax, int Dupire, PnlVect *row_stiffness);
extern void CIR_diffusion_update_time(CIR_diffusion *process, double t);
/*

```

Gamma- OU stochastic clock

the rate of time change is now solution of the SDE

$$dy_t = -\lambda y_t dt + dz_{\{\lambda t\}}$$

Choice z_t as a compound poisson process,

$$z_t = \sum_{n=1}^{N_t} x_n$$

where N_t is a Poisson process with intensity parameter α

and each x_n follows an exponential law with mean $\frac{1}{\beta}$.

*/

```

typedef struct _GammaOU_diffusion GammaOU_diffusion;

```

```

struct _GammaOU_diffusion

```

```

{

```

```

    double Lambda;

```

```

    double Alpha;

```

```

    double Beta;

```

```

    double y0;

```

```

    double Drift; // proportional to Drift correction

```

```

    double Lambda_a;

```

```

    double Lambda_b;

```

```

    double y0_one_m_el_div_lambda;

```

```

    double y0_el;

```

```

    double one_m_el_div_lambda;

```

```

    double beta_el;

```

```

    double time;
    double Jump_drift;
    double Jump_drift_psi;
    int nb_parameters;
    void *Levy;
    dcomplex(*characteristic_exponent)(dcomplex u, void *mod)
        ;
};

extern GammaOU_diffusion *GammaOU_diffusion_create(double
    Lambda, double Alpha, double Beta, double y0,
    void *Levy_,
    dcomplex(*characteristic_exponent_)(dcomplex, void *),
    double *jump_drift);
extern dcomplex GammaOU_diffusion_characteristic_exponent(
    dcomplex u, double t, void *mod);
extern dcomplex GammaOU_diffusion_ln_characteristic_
    function(dcomplex u, double t, void *mod);
extern double GammaOU_diffusion_get_sigma_square(GammaOU_
    diffusion *Process);
extern void GammaOU_diffusion_fourier_stiffness(GammaOU_diffusion *mod, double hx, double bnd_fourier, int Nw, int km
    in, int kmax, int Dupire, PnlVect *row_stiffness);
extern void GammaOU_diffusion_update_time(GammaOU_diffusion *process, double t);
extern void GammaOU_diffusion_list(const GammaOU_diffusion *process);

extern void test_CIR_diffusion(void);
extern void test_GammaOU_diffusion(void);

typedef struct _Levy_diffusion Levy_diffusion;

struct _Levy_diffusion
{
    void *process;
    int nb_parameters;
    int type_model;

```

```

    dcomplex(*characteristic_exponent)(dcomplex u, double t,
        void *mod);
    dcomplex(*ln_characteristic_function)(dcomplex u, double
        t, void *mod);
    // Artificial volatility term to come back to parabolic
    problem
    double vol_square;
};

extern Levy_diffusion *Levy_diffusion_create(void *process_
    , dcomplex(*characteristic_exponent_)(dcomplex u, double
    t, void *mod),
    dcomplex(*ln_characteristic_function_)(dcomplex u,
    double t, void *mod));
extern Levy_diffusion *Levy_diffusion_create_from_vect(int
    model, const double *input);
extern void Levy_diffusion_free(Levy_diffusion **Levy);
extern dcomplex Levy_diffusion_characteristic_exponent(dcom
    plex u, double t, Levy_diffusion *mod);
extern double Levy_diffusion_get_sigma_square(
    Levy_diffusion *Levy);
extern void Levy_diffusion_fourier_stiffness(
    Levy_diffusion *mod, double t, double hx, double bnd_fourier, int Nw,
    int kmin, int kmax, int Dupire, PnlVect *row_stiffness);
extern dcomplex Levy_diffusion_ln_characteristic_function(
    dcomplex u, double t, Levy_diffusion *mod);
extern dcomplex Levy_diffusion_ln_characteristic_function_w
    ith_cast(dcomplex u, double t, void *mod);
//extern dcomplex Levy_diffusion_characteristic_function(dc
    omplex u, double t, Levy_diffusion * mod);

extern void Levy_diffusion_constraints(PnlVect *res, const
    Levy_diffusion *Levy);

#endif

```

References