

[Help](#)

```
#ifndef __levy_process__
#define __levy_process__

#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 _BS_process BS_process;

struct _BS_process
{
    double sigma;
    double rate;
    int nb_parameters;
};
extern BS_process *BS_process_create(double sigma, double rate, double *jump_dri
extern BS_process *BS_process_create_from_vect(const PnlVect *input);
extern dcomplex BS_process_characteristic_exponent(dcomplex u, void *mod);
extern void BS_process_update_cast(void *process);

typedef struct _Merton_process Merton_process;

struct _Merton_process
{
    double sigma;
    double rate;
    double mu_J;
    double Sigma_J;
    double Lambda_J;
    double sigmaj_sqr_demi;
    double lnonepmuj;
    double Drift;
    int nb_parameters;
};
extern Merton_process *Merton_process_create(double sigma, double rate, double m
```

```

    double Lambda_J, double *jump_drift);

extern Merton_process *Merton_process_create_from_vect(const PnlVect *input);
extern dcomplex Merton_process_characteristic_exponent(dcomplex u, void *mod);
extern void Merton_process_update_cast(void *process);

typedef struct _CGMY_process CGMY_process;

struct _CGMY_process
{
    double C;
    double Y;
    double G;
    double M;
    // Artificial volatility term to come back to parabolic problem
    // Temporary variable, computed only one time
    double C_Gamma_minus_Y;
    double GpowY;
    double MpowY;
    double Gp1powY;
    double Mm1powY;
    double levyp;
    double levyn;
    double levynu;
    int nb_parameters;
};

extern CGMY_process *CGMY_process_create(double C, double G, double M, double Y,
extern CGMY_process *CGMY_process_create_from_vect(const PnlVect *input);
extern dcomplex CGMY_process_characteristic_exponent(dcomplex u, void *mod);
extern void CGMY_process_update_cast(void *process);

typedef struct _Temperedstable_process Temperedstable_process;

struct _Temperedstable_process
{
    double AlphaPlus;
    double AlphaMinus;
    double LambdaPlus;
    double LambdaMinus;

```

```

double CPlus;
double CMinus;
// Artificial volatility term to come back to parabolic problem
// Temporary variable, computed only one time
double C_Gamma_minus_Alpha_Plus;
double C_Gamma_minus_Alpha_Minus;
double LambdapowAlphaPlus;
double LambdapowAlphaMinus;
double Lambdam1powAlphaPlus;
double Lambdap1powAlphaMinus;
int nb_parameters;
};

extern Temperedstable_process *Temperedstable_process_create(double AlphaPlus, d
    double LambdaPlus, double LambdaMinus,
    double CPlus, double CMinus,
    double *jump_drift);
extern Temperedstable_process *Temperedstable_process_create_from_vect(const Pnl
extern dcomplex Temperedstable_process_characteristic_exponent(dcomplex u, void
extern void Temperedstable_process_characteristic_exponent_gradient(PnlVectCompl
extern void Temperedstable_process_update_cast(void *process);

typedef struct _NIG_process NIG_process;

struct _NIG_process
{
    double Alpha;
    double Beta;
    double Delta;

    //Second representation, use for Monte Carlo Simulation
    double Theta;
    double Sigma;
    double Nu;

    double Lambda; // proportional to Drift correction

    double Alpha_sqr;
    double Sqrt_Alpha2_minus_Beta2;

```

```

    int nb_parameters;
};

extern NIG_process *NIG_process_create(double Alpha, double Beta, double Delta,
extern NIG_process *NIG_process_create_from_vect(const PnlVect *input);
extern NIG_process *NIG_process_create_from_brownian_time(double sigma_, double
extern dcomplex NIG_process_characteristic_exponent(dcomplex u, void *mod);
extern void NIG_process_kill_drift(NIG_process *process);
extern void NIG_process_update_cast(void *process);

typedef struct _VG_process VG_process;

struct _VG_process
{
    double Kappa;
    double Theta;
    double Sigma;

    //Second representation, use for Monte Carlo Simulation
    double C;
    double G;
    double M;

    double Sigma_srq_demi;
    double Lambda; // proportional to Drift correction
    int nb_parameters;
};

extern VG_process *VG_process_create(double Kappa, double Theta, double Sigma, d
extern VG_process *VG_process_create_from_vect(const PnlVect *input);
extern VG_process *VG_process_create_from_CGM(double C, double G, double M, doub
extern dcomplex VG_process_characteristic_exponent(dcomplex u, void *mod);
extern void VG_process_kill_drift(VG_process *process);
extern void VG_process_update_cast(void *process);

/*

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_

```

$dW^1 dW^2 = \rho dt$

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

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

*/

```
typedef struct _Meixner_process Meixner_process;
```

```
struct _Meixner_process
```

```
{
```

```
    double Alpha;
```

```
    double Beta;
```

```
    double Delta;
```

```
    double cos_b2;
```

```
    double Lambda;
```

```
    // proportional to Drift correction d_ln_cos_b2_d_cos_apb2i
```

```
    int nb_parameters;
```

```
};
```

```
extern Meixner_process *Meixner_process_create(double Alpha, double Beta, double
```

```
extern Meixner_process *Meixner_process_create_from_vect(const PnlVect *input);
```

```
extern dcomplex Meixner_process_characteristic_exponent(dcomplex u, void *mod);
```

```
extern void Meixner_process_update_cast(void *process);
```

```
typedef struct _z_distribution_process z_distribution_process;
```

```
struct _z_distribution_process
```

```
{
```

```
    double Alpha;
```

```
    double Beta_1;
```

```
    double Beta_2;
```

```
    double Delta;
```

```
    double beta_b1_b2;
```

```
    double Lambda;
```

```
    // proportional to Drift correction d_ln_cos_b2_d_cos_apb2i
```

```

    int nb_parameters;

};

extern z_distribution_process *z_distribution_process_create(double Alpha, double
extern z_distribution_process *z_distribution_process_create_from_vect(const Pnl
extern dcomplex z_distribution_process_characteristic_exponent(dcomplex u, void
extern void z_distribution_process_update_cast(void *process);

typedef struct _Levy_process Levy_process;

struct _Levy_process
{
    int type_model;
    void *process;
    dcomplex(*characteristic_exponent)(dcomplex u, void *mod);
    void (*update)(void *process);
    int nb_parameters;
    // Artificial volatility term to come back to parabolic problem
    double vol_square;
    // Use for calibration, store the initial parameter of the model.
    double initial_parameter;
};

extern Levy_process *Levy_process_create(void *process_, int nb_parameters_, dco
extern Levy_process *Levy_process_create_from_vect(int model, const double *inpu
extern void Levy_process_free(Levy_process **);
extern dcomplex Levy_process_characteristic_exponent(dcomplex u, Levy_process *m
extern void Levy_process_update(Levy_process *mod);
extern double Levy_process_get_sigma_square(Levy_process *Levy);
extern void Levy_process_stiffness_by_fourier(Levy_process *mod, double hx, int
extern void Levy_process_stiffness_by_fourier_gradient(Levy_process *mod, double
extern dcomplex Levy_process_ln_characteristic_function(dcomplex u, double t, Le
extern dcomplex Levy_process_ln_characteristic_function_with_cast(dcomplex u, do
extern dcomplex Levy_process_characteristic_function(dcomplex u, double t, Levy_
extern double Levy_process_get_parameter(Levy_process *mod, int i);
extern void Levy_process_set_parameter(Levy_process *mod, int i, double v);
extern void Levy_process_shift_parameter(Levy_process *mod, int i, int sg, doubl
extern void Levy_process_restore_parameter(Levy_process *mod, int i);
extern void Levy_process_restore_parameter_without_restore(Levy_process *mod, in

```

```
extern void Levy_process_print_parameter(Levy_process *mod);  
// For calibration problem  
extern void Levy_process_constraints(PnlVect *res, const Levy_process *mod);  
  
#endif
```