Help

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <
href../../../common/math/cdo/cdo_math_h_src.pdfmath.h>
#include <assert.h>

#include "pnl/pnl_fft.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_band_matrix.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_mathtools.h"

#include "
href../../../common/math/equity_pricer/pde_tools_h_src.pdfpde_tools.h"
#include "
href../../../common/math/equity_pricer/levy_process_h_src.pdflevy_process.h"

#define IMPLICIT_VOL 0.0000
#define EPSILON_CALIBRATION 1e-2

#define GETPROCESSPARAMETER(v,i){              \
  if (i>=v->nb_parameters || i<0){           \
      perror("index out of range"); abort();}\
  else{return ((double *)v)[i];}}

#define SETPROCESSPARAMETER(v,i,a){   \
  if (i>=v->nb_parameters || i<0){           \
   perror("index out of range"); abort();}   \
  else{((double *)v)[i]=a;}}

#define GETLEVYPARAMETER(v,i){           \
  if (i>=v->nb_parameters || i<0){           \
      perror("index out of range"); abort();}\
  else{return ((double *)v->process)[i];}}

#define SETLEVYPARAMETER(v,i,a){  \
```

```
  if (i>=v->nb_parameters || i<0){           \
   perror("index out of range"); abort();}   \
  else{((double *)v->process)[i]=a;}}


/**
 * ln ( Gamma_ln (z+1)/z )
 * where Gamma_ln(z)=int_R^+ log(t) t^{z-1} exp(-t) dt,
 * the logarithm of the Gamma_log function
 * @param z  a complex number
 * @return ln (Gamma (z))
 */
dcomplex Clgamma_log(dcomplex z)
{
  dcomplex x, y, tmp, ser, sersq;
  static double cof[6] = {76.18009172947146, -86.50532032941677,
                          24.01409824083091, -1.231739572450155,
                          0.1208650973866179e-2, -0.5395239384953e-5
                         };
  double gamma = 5;
  int j;

  if (z.r < 0)
    PNL_ERROR("Error in Clngamma_log !", "Real part are not positive, CGMY.c ");
  y = x = z;
  tmp = Cadd(x, Complex(gamma + 0.5, 0));
  sersq = CZERO;
  ser = Complex(1.000000000190015, 0.0);

  for (j = 0; j <= 5; j++)
    {
      y = Cadd(y, CUNO);
      ser = Cadd(ser, Cdiv(Complex(cof[j], 0), y));
      sersq = Cadd(sersq, Cdiv(Complex(cof[j], 0), Cmul(y, y)));
    }

  ser = Cmul(Csub(Clog(tmp), Cdiv(Complex(gamma, 0), tmp)), ser);
  ser = Csub(ser, sersq);
  ser = RCmul(sqrt(M_2PI), ser);
  tmp = Csub(Cmul(Cadd(x, Complex(0.5, 0)), Clog(tmp)), tmp);
  return Cadd(Clog(Cdiv(ser, x)), tmp);
```

```
}


dcomplex Ctgamma_log(dcomplex z)
{
  if (z.r < 0)
    return Cdiv(Csub(Ctgamma_log(Complex(z.r + 1, 0)), Ctgamma(z)), z);
  return Csub(Cexp(Clgamma_log(z)), Cdiv(Cexp(Clgamma(z)), z));
}

double tbeta(double a, double b)
{
  return exp(pnl_lgamma(a) + pnl_lgamma(b) - pnl_lgamma(a + b));
}

dcomplex Ctbeta(dcomplex a, dcomplex b)
{
  return Cexp(Csub(Cadd(Clgamma(a), Clgamma(b)), Clgamma(Cadd(a, b))));
}

/*
dcomplex BS_characteristic_exponent(dcomplex u,BS_process * mod)
{
  // Test with BS process
  double vol_square =mod->vol;
  double r =mod->r;
  //>> Case 1 code infinitesimal generator of backward k=i-j
   return Complex(vol_square*u.r*u.r,-(r-vol_square)*u.r);
  //>> Case 2 code infinitesimal generator of diffuison k=j-i and use
  //>> bar(psi)(-u) = psi(u)
  //return Complex(vol_square*u.r*u.r,(r-vol_square)*u.r);
}
*/



static dcomplex M_minus_i_u_pow_Y_minus_M_pow_Y(dcomplex u, double Y, double M,
{

  double r = pow(M + u.i, 2) + u.r * u.r;
  // Problem with too small value of M, ?
```

```
  double theta = atan(-u.r / (M + u.i)) * Y;
  // Test of stabilized formula :
  // double x=-u.r/(M+u.i);
  // double theta = (fabs(x)<1)?atan(x):((x>0)?1:-1)*M_PI_2-atan(1./x);
  // theta *=Y;
  r = pow(r, Y / 2);
  return Complex(r * cos(theta) - MpowY, r * sin(theta));
}

static dcomplex characteristic_exponent_cgmy(dcomplex u, double G, double M, dou
{
  return Cadd(M_minus_i_u_pow_Y_minus_M_pow_Y(u, Y, M, MpowY),
              M_minus_i_u_pow_Y_minus_M_pow_Y(CRmul(u, -1), Y, G, GpowY));
}




// ----------------------------- BS ----------------------


dcomplex BS_process_characteristic_exponent_without_cast(dcomplex u, BS_process
{
  dcomplex psi = C_op_amib(RCmul(0.5 * mod->sigma, Cmul(u, u)), RCmul(mod->rate
  return CRadd(psi, mod->rate);
}

dcomplex BS_process_characteristic_exponent(dcomplex u, void *mod)
{
  return BS_process_characteristic_exponent_without_cast(u, (BS_process *) mod);
}

BS_process *BS_process_create(double sigma, double rate, double *jump_drift)
{
  BS_process *process = malloc(sizeof(BS_process));
  process->sigma = sigma;
  process->rate = rate;
  process->nb_parameters = 2;

  //>> Two way to compute drift term due to jump,
  //>> Put on Band matrix
```

```
  (*jump_drift) = 0;
  //>> Or Put in FD scheme (comment previous line and uncomment to next line)
  //  (*jump_drift)= -process->C_Gamma_minus_Alpha_Minus*process->Lambdap1powAlp
  //process->Lambdap1powAlphaMinus=0.0;
  return process;
};

BS_process *BS_process_create_from_vect(const PnlVect *input)
{
  int i;
  BS_process *process = malloc(sizeof(BS_process));
  process->nb_parameters = 2;
  for (i = 0; i < process->nb_parameters; i++)
    SETPROCESSPARAMETER(process, i, GET(input, i));
  return process;
};


void BS_process_update_cast(void *process)
{};


// ----------------------------- Merton -----------------------


dcomplex Merton_process_characteristic_exponent_without_cast(dcomplex u, Merton_
{
  dcomplex u_sqr_plus_i_u = Cmul(u, Complex(u.r, u.i + 1));
  dcomplex psi = C_op_amib(RCmul(0.5 * mod->sigma, u_sqr_plus_i_u), RCmul(mod->r
  /*
    dcomplex psi_J=RCmul(-mod->sigmaj_sqr_demi,u_sqr_plus_i_u);
    psi_J=C_op_apib(psi_J,RCmul(mod->lnonepmuj,u));
    psi_J=RCadd(-1,Cexp(psi_J));
  */
  dcomplex psi_J = C_op_apib(RCmul(-mod->sigmaj_sqr_demi, Cmul(u, u)), RCmul(mod
  psi_J = Csub(Cexp(psi_J), CUNO);
  psi = Csub(psi, RCmul(mod->Lambda_J, psi_J));
  psi = C_op_apib(psi, CRmul(u, mod->Drift));
  return CRadd(psi, mod->rate);
}
```

```
dcomplex Merton_process_characteristic_exponent(dcomplex u, void *mod)
{
  return Merton_process_characteristic_exponent_without_cast(u, (Merton_process
}
void Merton_process_update(Merton_process *process)
{
  process->sigmaj_sqr_demi = 0.5 * process->Sigma_J * process->Sigma_J;
  process->lnonepmuj = log(1 + process->mu_J);
  process->Drift = process->Lambda_J * (exp(process->mu_J + process->sigmaj_sqr_
}

Merton_process *Merton_process_create(double sigma, double rate,
                                      double mu_J_, double Sigma_J_, double Lamb

{
  Merton_process *process = malloc(sizeof(Merton_process));
  process->sigma = sigma;
  process->rate = rate;
  process->mu_J = mu_J_;
  process->Sigma_J = Sigma_J_;
  process->Lambda_J = Lambda_J_;
  process->nb_parameters = 5;
  //>> Two way to compute drift term due to jump,
  //>> Put on Band matrix
  (*jump_drift) = 0;
  //>> Or Put in FD scheme (comment previous line and uncomment to next line)
  //  (*jump_drift)= -process->C_Gamma_minus_Alpha_Minus*process->Lambdap1powAlp
  //process->Lambdap1powAlphaMinus=0.0;
  return process;
};

Merton_process *Merton_process_create_from_vect(const PnlVect *input)
{
  int i;
  Merton_process *process = malloc(sizeof(Merton_process));
  process->nb_parameters = 5;
  for (i = 0; i < process->nb_parameters; i++)
    SETPROCESSPARAMETER(process, i, GET(input, i));
  Merton_process_update(process);
  return process;
};
```

```
void Merton_process_update_cast(void *process)
{
  Merton_process_update((Merton_process *)process);
}

// ----------------------------- CGMY ----------------------

dcomplex CGMY_process_characteristic_exponent_without_cast(dcomplex u, CGMY_proc
{

  //>> To add special cas Y=0 and Y=1 (Gamma not defined )

  //>> Case 1 code infinitesimal generator of backward k=i-j
  dcomplex psi = characteristic_exponent_cgmy(u, mod->G, mod->M, mod->Y, mod->Mp
  psi = C_op_amib(psi, CRmul(u, mod->Mm1powY));
  return RCmul(-mod->C_Gamma_minus_Y, psi);
  //>> Case 2 code infinitesimal generator of diffuison k=j-i and use
  //>> bar(psi)(-u) = psi(u)
  /*
    To do
    dcomplex psi =characteristic_exponent_cgmy(u,mod->G,mod->M,mod->Y,mod->MpowY
    psi=C_op_amib(psi,CRmul(u,mod->Mm1powY));
    return RCmul(-mod->C_Gamma_minus_Y,psi);
  */

}

dcomplex CGMY_process_characteristic_exponent(dcomplex u, void *mod)
{
  return CGMY_process_characteristic_exponent_without_cast(u, (CGMY_process *) m
}

void CGMY_process_update(CGMY_process *process)
{
  process->GpowY = pow(process->G, process->Y);
  process->MpowY = pow(process->M, process->Y);
  process->C_Gamma_minus_Y = process->C * pnl_tgamma(-process->Y);
  process->Gp1powY = pow(process->G + 1, process->Y) - process->GpowY;
```

```
  process->Mm1powY = pow(process->M - 1, process->Y) - process->MpowY + process-
}

CGMY_process *CGMY_process_create(double C, double G, double M, double Y, double
{
  CGMY_process *process = malloc(sizeof(CGMY_process));
  process->C = C;
  process->G = G;
  process->M = M;
  process->Y = Y;
  process->nb_parameters = 4;
  CGMY_process_update(process);
  //printf(" Jump drift correction plus %7.4f \ n",-process->C_Gamma_minus_Y*pro
  //printf(" Jump drift correction %7.4f \ n",-process->C_Gamma_minus_Y*process-
  //>> Two way to compute drift term due to jump,
  //>> Put on Band matrix
  (*jump_drift) = 0;
  //>> Or Put in FD scheme (comment previous line and uncomment to next line)
  //(*jump_drift)= -process->C_Gamma_minus_Y*process->Mm1powY;
  //process->C_Gamma_minus_Y*process->Mm1powY=0;

  process->levyp = process->M;
  process->levyn = process->G;
  process->levynu = 1.;
  return process;
};

CGMY_process *CGMY_process_create_from_vect(const PnlVect *input)
{
  int i;
  CGMY_process *process = malloc(sizeof(CGMY_process));
  process->nb_parameters = 4;
  for (i = 0; i < process->nb_parameters; i++)
    SETPROCESSPARAMETER(process, i, GET(input, i));
  CGMY_process_update(process);
  return process;
};


void CGMY_process_update_cast(void *process)
{
```

```
  CGMY_process_update((CGMY_process *)process);
}

// ----------------------------- Temperedstable -----------------------

dcomplex Temperedstable_process_characteristic_exponent_without_cast(dcomplex u,
{
  //>> Case 1 code infinitesimal generator of backward k=i-j
  dcomplex psiplus  = M_minus_i_u_pow_Y_minus_M_pow_Y(u, mod->AlphaPlus, mod->La
  dcomplex psiminus = M_minus_i_u_pow_Y_minus_M_pow_Y(Complex(-u.r, -u.i), mod->
  psiplus = C_op_amib(psiplus, CRmul(u, mod->Lambdam1powAlphaPlus));
  psiminus = C_op_amib(psiminus, CRmul(u, mod->Lambdap1powAlphaMinus));
  psiplus = Cadd(RCmul(-mod->C_Gamma_minus_Alpha_Plus, psiplus), RCmul(-mod->C_G
  // Now substract implicite diffusion term
  return psiplus;
  //>> Case 2 code infinitesimal generator of diffuison k=j-i and use
  //>> bar(psi)(-u) = psi(u)
  //  To do
}

dcomplex Temperedstable_process_characteristic_exponent(dcomplex u, void *mod)
{
  return Temperedstable_process_characteristic_exponent_without_cast(u, (Tempere
}

void Temperedstable_process_update(Temperedstable_process *process)
{
  process->LambdapowAlphaPlus = pow(process->LambdaPlus, process->AlphaPlus);
  process->LambdapowAlphaMinus = pow(process->LambdaMinus, process->AlphaMinus);
  process->Lambdam1powAlphaPlus = pow(process->LambdaPlus - 1.0, process->AlphaP
  process->Lambdap1powAlphaMinus = pow(process->LambdaMinus + 1.0, process->Alph
  process->C_Gamma_minus_Alpha_Plus = process->CPlus * pnl_tgamma(-process->Alph
  process->C_Gamma_minus_Alpha_Minus = process->CMinus * pnl_tgamma(-process->Al
}

Temperedstable_process *Temperedstable_process_create(double AlphaPlus, double A
    double LambdaMinus, double CPlus, double CMinus, double *jump_drift)
{
  Temperedstable_process *process = malloc(sizeof(Temperedstable_process));
  process->AlphaPlus = AlphaPlus;
  process->AlphaMinus = AlphaMinus;
```

```
  process->LambdaPlus = LambdaPlus;
  process->LambdaMinus = LambdaMinus;
  process->CPlus = CPlus;
  process->CMinus = CMinus;
  process->nb_parameters = 6;
  Temperedstable_process_update(process);
  //printf(" Jump drift correction Plus %7.4f \ n", -process->C_Gamma_minus_Alph
  //printf(" Jump drift correction Minus %7.4f \ n",-process->C_Gamma_minus_Alph
  //>> Two way to compute drift term due to jump,
  //>> Put on Band matrix
  (*jump_drift) = 0.; //-(process->C_Gamma_minus_Alpha_Plus*process->Lambdam1pow
  //process->Lambdam1powAlphaPlus=0;
  //process->Lambdap1powAlphaMinus=0;
  //>> Or Put in FD scheme (comment previous line and uncomment to next line)
  //  (*jump_drift)= -process->C_Gamma_minus_Alpha_Minus*process->Lambdap1powAlp
  //process->Lambdap1powAlphaMinus=0.0;

  return process;
};

Temperedstable_process *Temperedstable_process_create_from_vect(const PnlVect *i
{
  int i;
  Temperedstable_process *process = malloc(sizeof(Temperedstable_process));
  process->nb_parameters = 6;
  for (i = 0; i < process->nb_parameters; i++)
    SETPROCESSPARAMETER(process, i, GET(input, i));
  Temperedstable_process_update(process);
  return process;
};


void Temperedstable_process_update_cast(void *process)
{
  Temperedstable_process_update((Temperedstable_process *)process);
}

// ---------------------------- NIG ----------------------

dcomplex NIG_process_characteristic_exponent_without_cast(dcomplex u, NIG_proces
{
```

```
  dcomplex psi  = C_op_apib(Complex(mod->Beta, 0), u);
  psi = Cmul(psi, psi);
  psi = CRsub(Csqrt(RCsub(mod->Alpha_sqr, psi)), mod->Sqrt_Alpha2_minus_Beta2);
  psi = RCmul(mod->Delta, C_op_amib(psi, RCmul(mod->Lambda, u)));
  return psi;
}

dcomplex NIG_process_characteristic_exponent(dcomplex u, void *mod)
{
  return NIG_process_characteristic_exponent_without_cast(u, (NIG_process *) mod
}

void NIG_process_update(NIG_process *process)
{
  process->Sigma = process->Delta;
  process->Theta = process->Beta * process->Delta * process->Delta;
  process->Nu = sqrt(process->Alpha * process->Alpha - process->Beta * process->

  process->Alpha_sqr = process->Alpha * process->Alpha;
  process->Sqrt_Alpha2_minus_Beta2 = sqrt(process->Alpha_sqr - process->Beta * p
  process->Lambda = sqrt(process->Alpha_sqr - pow(process->Beta + 1, 2)) - proce

}
NIG_process *NIG_process_create(double Alpha, double Beta, double Delta, double
{
  NIG_process *process = malloc(sizeof(NIG_process));
  process->Alpha = Alpha;
  process->Beta = Beta;
  process->Delta = Delta;
  process->nb_parameters = 3;
  NIG_process_update(process);

  //>> Two way to compute drift term due to jump,
  //>> Put on Band matrix
  (*jump_drift) = 0;
  //>> Or Put in FD scheme (comment previous line and uncomment to next line)
  //  (*jump_drift)= -process->C_Gamma_minus_Alpha_Minus*process->Lambdap1powAlp
  //process->Lambdap1powAlphaMinus=0.0;
  return process;
};
```

```
NIG_process *NIG_process_create_from_vect(const PnlVect *input)
{
  int i;
  NIG_process *process = malloc(sizeof(NIG_process));
  process->nb_parameters = 3;
  for (i = 0; i < process->nb_parameters; i++)
    SETPROCESSPARAMETER(process, i, GET(input, i));
  NIG_process_update(process);
  return process;
};


NIG_process *NIG_process_create_from_brownian_time(double sigma_, double nu_, do
{
  double sigma_sqr = sigma_ * sigma_;
  NIG_process *process = malloc(sizeof(NIG_process));
  process->Theta = theta_;
  process->Sigma = sigma_;
  process->Nu = nu_;
  process->nb_parameters = 3;

  process->Alpha = sqrt(nu_ * nu_ / sigma_sqr + theta_ * theta_ / (sigma_sqr * s
  process->Beta = theta_ / (sigma_sqr);
  process->Delta = sigma_;
  NIG_process_update(process);
  //>> Two way to compute drift term due to jump,
  //>> Put on Band matrix
  (*jump_drift) = 0;
  //>> Or Put in FD scheme (comment previous line and uncomment to next line)
  //  (*jump_drift)= -process->C_Gamma_minus_Alpha_Minus*process->Lambdap1powAlp
  //process->Lambdap1powAlphaMinus=0.0;
  return process;
};

void NIG_process_kill_drift(NIG_process   *process)
{
  process->Lambda = 0.0;
};
void NIG_process_update_cast(void *process)
{
  NIG_process_update((NIG_process *)process);
```

```
}

// ----------------------------- VG -----------------------

dcomplex VG_process_characteristic_exponent_without_cast(dcomplex u, VG_process
{
  //>> Case 1 code infinitesimal generator of backward k=i-j
  dcomplex psi  = RCmul(mod->Kappa, C_op_amib(RCmul(mod->Sigma_srq_demi, Cmul(u,
  return RCmul(1 / mod->Kappa, C_op_amib(Clog(RCadd(1, psi)), RCmul(mod->Lambda,
}


dcomplex VG_process_characteristic_exponent(dcomplex u, void *mod)
{
  return VG_process_characteristic_exponent_without_cast(u, (VG_process *) mod);
}

void VG_process_update(VG_process *process)
{
  process->C = 1. / process->Kappa;
  process->G = sqrt(0.25 * process->Theta * process->Theta * process->Kappa * pr
  process->M = 1.0 / (process->G + 0.5 * process->Theta * process->Kappa);
  process->G = 1.0 / (process->G - 0.5 * process->Theta * process->Kappa);

  process->Sigma_srq_demi = process->Sigma * process->Sigma * 0.5;
  process->Lambda = log(1 - process->Kappa * (process->Sigma_srq_demi + process-
}
VG_process *VG_process_create(double Kappa, double Theta, double Sigma, double *
{
  VG_process *process = malloc(sizeof(VG_process));
  process->Kappa = Kappa;
  process->Theta = Theta;
  process->Sigma = Sigma;
  process->nb_parameters = 3;
  VG_process_update(process);
  //>> Two way to compute drift term due to jump,
  //>> Put on Band matrix
  (*jump_drift) = 0;
  //>> Or Put in FD scheme (comment previous line and uncomment to next line)
  //  (*jump_drift)= -process->C_Gamma_minus_Alpha_Minus*process->Lambdap1powAlp
  //process->Lambdap1powAlphaMinus=0.0;
```

```
  return process;
};

VG_process *VG_process_create_from_vect(const PnlVect *input)
{
  int i;
  VG_process *process = malloc(sizeof(VG_process));
  process->nb_parameters = 3;
  for (i = 0; i < process->nb_parameters; i++)
    SETPROCESSPARAMETER(process, i, GET(input, i));
  VG_process_update(process);
  return process;
};


VG_process *VG_process_create_from_CGM(double C, double G, double M, double *jum
{
  VG_process *process = malloc(sizeof(VG_process));
  process->C = C;
  process->G = G;
  process->M = M;
  process->nb_parameters = 3;

  process->Kappa = 1.0 / C;
  process->Theta = C / M - C / G;
  process->Sigma = sqrt(2 * C / (G * M));
  VG_process_update(process);
  //>> Two way to compute drift term due to jump,
  //>> Put on Band matrix
  (*jump_drift) = 0;
  //>> Or Put in FD scheme (comment previous line and uncomment to next line)
  //  (*jump_drift)= -process->C_Gamma_minus_Alpha_Minus*process->Lambdap1powAlp
  //process->Lambdap1powAlphaMinus=0.0;
  return process;
};

void VG_process_kill_drift(VG_process   *process)
{
  process->Lambda = 0.0;
};
```

```
void VG_process_update_cast(void *process)
{
  VG_process_update((VG_process *)process);
}

// ---------------------------- Meixner ----------------------

dcomplex Meixner_process_characteristic_exponent_without_cast(dcomplex u, Meixne
{
  //>> Case 1 code infinitesimal generator of backward k=i-j
  dcomplex psi  = RCmul(0.5 * mod->Alpha, u);
  psi = Complex(psi.r, psi.i - 0.5 * mod->Beta);
  psi = Clog(RCmul(mod->cos_b2, Ccosh(psi)));
  return C_op_amib(RCmul(2 * mod->Delta, psi), RCmul(mod->Lambda, u));
}


dcomplex Meixner_process_characteristic_exponent(dcomplex u, void *mod)
{
  return Meixner_process_characteristic_exponent_without_cast(u, (Meixner_proces
}

void Meixner_process_update(Meixner_process *process)
{
  process->cos_b2 = 1.0 / cos(process->Beta * 0.5);
  process->Lambda = 2 * process->Delta * log((cos(0.5 * (process->Alpha + proces
}

Meixner_process *Meixner_process_create(double Alpha, double Beta, double Delta,
{
  Meixner_process *process = malloc(sizeof(Meixner_process));
  process->Alpha = Alpha;
  process->Beta = Beta;
  process->Delta = Delta;
  process->nb_parameters = 3;
  Meixner_process_update(process);
  (*jump_drift) = 0;
  return process;
};

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

```
{
  int i;
  Meixner_process *process = malloc(sizeof(Meixner_process));
  process->nb_parameters = 3;
  for (i = 0; i < process->nb_parameters; i++)
    SETPROCESSPARAMETER(process, i, GET(input, i));
  Meixner_process_update(process);
  return process;
};

void Meixner_process_update_cast(void *process)
{
  Meixner_process_update((Meixner_process *)process);
}


// ----------------------------- z_distribution -----------------------

dcomplex z_distribution_process_characteristic_exponent_without_cast(dcomplex u,
{
  //>> Case 1 code infinitesimal generator of backward k=i-j
  //>> Case 1 code infinitesimal generator of backward k=i-j
  dcomplex psi   = RCmul(0.5 * mod->Alpha, u);
  dcomplex psi2  = Complex(mod->Beta_2 - psi.i, psi.r);
  psi = Complex(mod->Beta_1 - psi.i, psi.r);
  psi = Clog(RCmul(mod->beta_b1_b2, Ctbeta(psi, psi2)));
  return C_op_amib(RCmul(-2 * mod->Delta, psi), RCmul(mod->Lambda, u));
}

dcomplex z_distribution_process_characteristic_exponent(dcomplex u, void *mod)
{
  return z_distribution_process_characteristic_exponent_without_cast(u, (z_distr
}


void z_distribution_process_update(z_distribution_process *process)
{
  process->beta_b1_b2 = 1.0 / tbeta(process->Beta_1, process->Beta_2);
  process->Lambda = 2 * process->Delta * log(tbeta(process->Beta_1 + 0.5 * M_1_P
}
```

```
z_distribution_process *z_distribution_process_create(double Alpha, double Beta_
{
  z_distribution_process *process = malloc(sizeof(z_distribution_process));
  process->Alpha = Alpha;
  process->Beta_1 = Beta_1;
  process->Beta_2 = Beta_2;
  process->Delta = Delta;
  process->nb_parameters = 4;
  z_distribution_process_update(process);
  (*jump_drift) = 0;
  return process;
};


z_distribution_process *z_distribution_process_create_from_vect(const PnlVect *i
{
  int i;
  z_distribution_process *process = malloc(sizeof(z_distribution_process));
  process->nb_parameters = 3;
  for (i = 0; i < process->nb_parameters; i++)
    SETPROCESSPARAMETER(process, i, GET(input, i));
  z_distribution_process_update(process);
  return process;
};

void z_distribution_process_update_cast(void *process)
{
  z_distribution_process_update((z_distribution_process *)process);
}


// ---------------  Levy --------------------
dcomplex Levy_process_characteristic_exponent(dcomplex u, Levy_process *mod)
{
  return Csub(mod->characteristic_exponent(u, mod->process),
              Complex(mod->vol_square * (u.r * u.r - u.i * u.i + u.i), (mod->vol
}

dcomplex Levy_process_ln_characteristic_function(dcomplex u, double t, Levy_proc
{
  return RCmul(-t, Levy_process_characteristic_exponent(u, mod));
```

```
}

dcomplex Levy_process_ln_characteristic_function_with_cast(dcomplex u, double t,
{
  return RCmul(-t, Levy_process_characteristic_exponent(u, (Levy_process *)mod))
}

dcomplex Levy_process_characteristic_function(dcomplex u, double t, Levy_process
{
  return Cexp(Levy_process_ln_characteristic_function(u, t, mod));
}


double Levy_process_get_sigma_square(Levy_process *Levy)
{
  return Levy->vol_square;
};


Levy_process *Levy_process_create(void *process_,
                                  int nb_parameters_,
                                  dcomplex(*characretristic_exponent_)(dcomplex
{
  Levy_process *Levy = malloc(sizeof(Levy_process));
  Levy->process = process_;
  Levy->nb_parameters = nb_parameters_;
  Levy->characteristic_exponent = characretristic_exponent_;
  Levy->update = update_;
  Levy->vol_square = IMPLICIT_VOL;
  return Levy;
};


Levy_process *Levy_process_create_from_vect(int model, const double *input)
{
  Levy_process *Levy = malloc(sizeof(Levy_process));
  PnlVect input_v;
  Levy->type_model = model;
  switch (model)
    {
    case 1:
```

```
  input_v = pnl_vect_wrap_array(input, 2);
  Levy->process = (void *)BS_process_create_from_vect(&input_v);
  Levy->nb_parameters = ((BS_process *) Levy->process)->nb_parameters;
  Levy->characteristic_exponent = BS_process_characteristic_exponent;
  break;
case 2:
  input_v = pnl_vect_wrap_array(input, 5);
  Levy->process = (void *)Merton_process_create_from_vect(&input_v);
  Levy->nb_parameters = ((Merton_process *) Levy->process)->nb_parameters;
  Levy->characteristic_exponent = Merton_process_characteristic_exponent;
  break;
case 3:
  input_v = pnl_vect_wrap_array(input, 4);
  Levy->process = (void *)CGMY_process_create_from_vect(&input_v);
  Levy->nb_parameters = ((CGMY_process *) Levy->process)->nb_parameters;
  Levy->characteristic_exponent = CGMY_process_characteristic_exponent;
  break;
case 4:
  input_v = pnl_vect_wrap_array(input, 6);
  Levy->process = (void *)Temperedstable_process_create_from_vect(&input_v);
  Levy->nb_parameters = ((Temperedstable_process *) Levy->process)->nb_param
  Levy->characteristic_exponent = Temperedstable_process_characteristic_expo
  break;
case 5:
  input_v = pnl_vect_wrap_array(input, 3);
  Levy->process = (void *)VG_process_create_from_vect(&input_v);
  Levy->nb_parameters = ((VG_process *) Levy->process)->nb_parameters;
  Levy->characteristic_exponent = VG_process_characteristic_exponent;
  break;
case 6:
  input_v = pnl_vect_wrap_array(input, 3);
  Levy->process = (void *)NIG_process_create_from_vect(&input_v);
  Levy->nb_parameters = ((NIG_process *) Levy->process)->nb_parameters;
  Levy->characteristic_exponent = NIG_process_characteristic_exponent;
  break;
case 7:
  input_v = pnl_vect_wrap_array(input, 3);
  Levy->process = (void *)Meixner_process_create_from_vect(&input_v);
  Levy->nb_parameters = ((Meixner_process *) Levy->process)->nb_parameters;
  Levy->characteristic_exponent = Meixner_process_characteristic_exponent;
  break;
```

```
    case 8:
      input_v = pnl_vect_wrap_array(input, 4);
      Levy->process = (void *)z_distribution_process_create_from_vect(&input_v);
      Levy->nb_parameters = ((z_distribution_process *) Levy->process)->nb_param
      Levy->characteristic_exponent = z_distribution_process_characteristic_expo
      break;
    default:
      return NULL;
    }
  Levy->vol_square = IMPLICIT_VOL;
  return Levy;
}

void Levy_process_free(Levy_process **Levy)
{
  switch ((*Levy)->type_model)
    {
    case 1:
      free((BS_process *)((*Levy)->process));
      break;
    case 2:
      free((BS_process *)((*Levy)->process));
      break;
    case 3:
      free((CGMY_process *)((*Levy)->process));
      break;
    case 4:
      free((Temperedstable_process *)((*Levy)->process));
      break;
    case 5:
      free((VG_process *)((*Levy)->process));
      break;
    case 6:
      free((NIG_process *)((*Levy)->process));
      break;
    case 7:
      free((Meixner_process *)((*Levy)->process));
      break;
    case 8:
      free((z_distribution_process *)((*Levy)->process));
      break;
```

```
    default:
    {
      ;
    }
    }
  free(*Levy);
  *Levy = NULL;
};

void Levy_process_update(Levy_process *mod)
{
  mod->update(mod->process);
};

double Levy_process_get_parameter(Levy_process *mod, int i)
{
  GETLEVYPARAMETER(mod, i);
}

void Levy_process_set_parameter(Levy_process *mod, int i, double v)
{
  SETLEVYPARAMETER(mod, i, v);
}

void Levy_process_shift_parameter(Levy_process *mod, int i, int sg, double *shif
{
  mod->initial_parameter = Levy_process_get_parameter(mod, i);
  ((double *)mod->process)[i] += sg * EPSILON_CALIBRATION; //*=(1+sg*EPSILON_CAL
  *shifted = EPSILON_CALIBRATION; //mod->initial_parameter*EPSILON_CALIBRATION;
  *shifted *= 2;
  Levy_process_update(mod);
}

void Levy_process_restore_parameter_without_restore(Levy_process *mod, int i)
{
  SETLEVYPARAMETER(mod, i, mod->initial_parameter);
}
void Levy_process_restore_parameter(Levy_process *mod, int i)
{
  Levy_process_restore_parameter_without_restore(mod, i);
  Levy_process_update(mod);
```

```
}

void Levy_process_print_parameter(Levy_process *mod)
{
  int i;
  for (i = 0; i < mod->nb_parameters; i++)
    printf("%7.4f, ", ((double *)mod->process)[i]);
  printf("\ n");
}


#undef GETPROCESSPARAMETER
#undef SETPROCESSPARAMETER
#undef GETLEVYPARAMETER
#undef SETLEVYPARAMETER
```