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_integration.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_complex.h"
#include "
href../../../common/math/equity_pricer/levy_diffusion_h_src.pdflevy_diffusion.h"
#include "
href../../../common/math/equity_pricer/carr_h_src.pdfcarr.h"



// ----------------- Attari method -------------------------
double AttariEvaluation(double w,
                        double T,
                        double l,
                        void *Model,
                        dcomplex(*ln_phi)(dcomplex u, double t, void *model))
{
  dcomplex Phi = Cexp(ln_phi(Complex(w, 0), T, Model));
  double   Re  = Creal(Phi);
  double   Im  = Cimag(Phi);
  return (1 / (1 + w * w)) * ((Re + Im / w) * cos(w * l) + (Im - Re / w) * sin(w
}

double AttariEvaluation_derivative(double w,
                                   double T,
                                   double l,
                                   void *Model,
                                   dcomplex(*ln_phi)(dcomplex u, double t, void
{
  dcomplex Phi = Cexp(ln_phi(Complex(w, 0), T, Model));
  double   Re  = Creal(Phi);
```

```
  double   Im  = Cimag(Phi);
  return ((1 / (1 + w * w)) * (-1.0 * (w * Re + Im) * sin(w * l) + (w * Im - Re)
}

typedef struct
{
  double T;
  double l;
  void *Model;
  dcomplex(*ln_phi)(dcomplex u, double t, void *model);
} AttariFunc ;

AttariFunc *Attari_func_create(double T, double l, void *Model,
                               dcomplex(*ln_phi)(dcomplex u, double t, void *mod
{
  AttariFunc *params = malloc(sizeof(AttariFunc));
  params->T = T;
  params->l = l;
  params->Model = Model;
  params->ln_phi = ln_phi;
  return params;
}

double AttariEvaluation_Obj(double w,
                            AttariFunc *Obj)
{
  return AttariEvaluation(w, Obj->T, Obj->l, Obj->Model, Obj->ln_phi);
}

double AttariEvaluation_Void(double w,
                             void *Obj)
{
  return AttariEvaluation_Obj(w, (AttariFunc *)Obj);
}

double AttariEvaluation_derivative_Obj(double w,
                                       AttariFunc *Obj)
{
  return AttariEvaluation_derivative(w, Obj->T, Obj->l, Obj->Model, Obj->ln_phi)
}
```

```
double AttariEvaluation_derivative_void(double w,
                                        void *Obj)
{
  return AttariEvaluation_derivative_Obj(w, (AttariFunc *)Obj);
}



int AttariMethod(double S0,
                 double T,
                 double K,
                 double CallPut,
                 double r,
                 double divid,
                 double sigma,
                 void *Model,
                 dcomplex(*ln_phi)(dcomplex u, double t, void *model),
                 double *ptprice,
                 double *ptdelta)
{

  // We only want the part corresponding to the driving stochastic process and
  // not the part containing the forward price (in Attari's formulation)
  // Since above we have normalized everything with respect to S0, we use
  // spot = 1 here

  // In this formulation, the FW and ZC enters below;
  // this is the char fun of the log of the martingale part of the spot diffusio
  PnlFunc Func;
  double InverseFourier, abserr;
  int neval;
  AttariFunc *attari_function;
  double epsabs = 1e-5;
  double epsrel = 1e-8;
  double A = 100;
  double l = log(K / S0) - (r - divid) * T;
  attari_function = Attari_func_create(T, l, Model, ln_phi);
  Func.params = attari_function;
  Func.F = &AttariEvaluation_Void;
  pnl_integration_GK(&Func, 0, A, epsabs, epsrel, &InverseFourier, &abserr, &nev
  *ptprice = S0 * exp(-divid * T) * ((CallPut == 1) ? (1.0 - exp(l) * (0.5 + M_1
```

```
  Func.F = &AttariEvaluation_derivative_void;
  pnl_integration_GK(&Func, 0, A, epsabs, epsrel, &InverseFourier, &abserr, &nev
  *ptdelta = exp(-divid * T) * (((CallPut == 1) ? 1.0 : 0.0) + exp(l) * M_1_PI *
  //memory desallocation
  //memory desallocation
  free(attari_function);
  return OK;
}
```

```
// ---------------- Attari method on [0,1] ------------------------
double AttariEvaluation01(double u,
                          double T,
                          double l,
                          void *Model,
                          dcomplex(*ln_phi)(dcomplex u, double t, void *model)
                         )
{
  double w = u / (1 - u);
  dcomplex Phi = Cexp(ln_phi(Complex(w, 0), T, Model));
  double   Re  = Creal(Phi);
  double   Im  = Cimag(Phi);
  return 1 / pow(1 - u, 2) * ((1) / (1 + w * w)) * ((Re + Im / w) * cos(w * l) +
  /*
  double res,lr;
  int p,lp= floor(l);
  lr=l-lp;
  p = floor(w/M_2PI);
  res= w-p*M_2PI;
  return   1/pow(1-u,2)*((1/(1 + w*w))*((Re +Im/w)*cos(lp*res+lr*w)+(Im - Re/w)*
  */
}

double AttariEvaluation01_derivative(double u,
                                     double T,
                                     double l,
                                     void *Model,
                                     dcomplex(*ln_phi)(dcomplex u, double t, voi
```

```
{
  double w = u / (1 - u);
  dcomplex Phi = Cexp(ln_phi(Complex(w, 0), T, Model));
  double   Re  = Creal(Phi);
  double   Im  = Cimag(Phi);
  return   1 / pow(1 - u, 2) * ((1 / (1 + w * w)) * (-1.0 * (w * Re + Im) * sin(
  /*
    double res,lr;
    int p,lp= floor(l);
    lr=l-lp;
    p = floor(w/M_2PI);
    res= w-p*M_2PI;
    return   1/pow(1-u,2)*((1/(1 + w*w))*(-1.0*(w*Re +Im)*sin(lp*res+lr*w)+(w*Im
  */
}

double AttariEvaluation_Obj01(double w,
                              AttariFunc *Obj)
{
  return AttariEvaluation01(w, Obj->T, Obj->l, Obj->Model, Obj->ln_phi);
}

double AttariEvaluation_Void01(double w,
                               void *Obj)
{
  return AttariEvaluation_Obj01(w, (AttariFunc *)Obj);
}


double AttariEvaluation_derivative_Obj01(double w,
    AttariFunc *Obj)
{
  return AttariEvaluation01_derivative(w, Obj->T, Obj->l, Obj->Model, Obj->ln_ph
}

double AttariEvaluation_derivative_Void01(double w,
    void *Obj)
{
  return AttariEvaluation_derivative_Obj01(w, (AttariFunc *)Obj);
}
```

```
int AttariMethod_on01_price(double S0,
                            double T,
                            double K,
                            double CallPut,
                            double r,
                            double divid,
                            double sigma,
                            void *Model,
                            dcomplex(*ln_phi)(dcomplex u, double t, void *model)
                            double *ptprice)
{

  // We only want the part corresponding to the driving stochastic process and
  // not the part containing the forward price (in Attari's formulation)
  // Since above we have normalized everything with respect to S0, we use
  // spot = 1 here
  // In this formulation, the FW and ZC enters below;
  // this is the char fun of the log of the martingale part of the spot diffusio
  PnlFunc Func;
  double InverseFourier, abserr;
  int neval;
  AttariFunc *attari_function;
  double epsabs = 1e-5;
  double epsrel = 1e-8;
  double l = log(K / S0) - (r - divid) * T;
  attari_function = Attari_func_create(T, l, Model, ln_phi);
  Func.params = attari_function;
  Func.F = &AttariEvaluation_Void01;
  pnl_integration_GK(&Func, 0, 1, epsabs, epsrel, &InverseFourier, &abserr, &nev
  *ptprice = S0 * exp(-divid * T) * ((CallPut == 1) ? (1.0 - exp(l) * (0.5 + M_1
  free(attari_function);
  return OK;
}


int AttariMethod_on01(double S0,
                      double T,
                      double K,
                      double CallPut,
                      double r,
```

```
                      double divid,
                      double sigma,
                      void *Model,
                      dcomplex(*ln_phi)(dcomplex u, double t, void *model),
                      double *ptprice,
                      double *ptdelta)
{

  // We only want the part corresponding to the driving stochastic process and
  // not the part containing the forward price (in Attari's formulation)
  // Since above we have normalized everything with respect to S0, we use
  // spot = 1 here
  // In this formulation, the FW and ZC enters below;
  // this is the char fun of the log of the martingale part of the spot diffusio
  PnlFunc Func;
  double InverseFourier, abserr;
  int neval;
  AttariFunc *attari_function;
  double epsabs = 1e-5;
  double epsrel = 1e-8;
  double l = log(K / S0) - (r - divid) * T;
  attari_function = Attari_func_create(T, l, Model, ln_phi);
  Func.params = attari_function;
  Func.F = &AttariEvaluation_Void01;
  pnl_integration_GK(&Func, 0, 0.99, epsabs, epsrel, &InverseFourier, &abserr, &
  *ptprice = S0 * exp(-divid * T) * ((CallPut == 1) ? (1.0 - exp(l) * (0.5 + M_1
  Func.F = &AttariEvaluation_derivative_Void01;
  pnl_integration_GK(&Func, 0, 0.99, epsabs, epsrel, &InverseFourier, &abserr, &
  *ptdelta = exp(-divid * T) * (((CallPut == 1) ? 1.0 : 0.0) + exp(l) * M_1_PI *
  //Put Case via parity*/
  //memory desallocation
  free(attari_function);
  return OK;
}



int AttariMethod_Vanilla_option(Option_Eqd *opt,
                                double sigma,
                                Levy_process *Model)
{
```

```
  if (opt->product_type != 1)
    PNL_ERROR(" Attari method works only for european option !", "attari.c ");
  return AttariMethod(opt->S0, opt->T, opt->K, opt->product, opt->rate, opt->div
                      &Levy_process_ln_characteristic_function_with_cast, &(opt-
}


int AttariMethod_Vanilla_option_LD(Option_Eqd *opt,
                                   double sigma,
                                   Levy_diffusion *Model)
{
  if (opt->product_type != 1)
    PNL_ERROR(" Attari method works only for european option !", "attari.c ");

  return AttariMethod(opt->S0, opt->T, opt->K, opt->product, opt->rate, opt->div
                      &Levy_diffusion_ln_characteristic_function_with_cast, &(op
}
```