Help

```
#include <stdio.h>
#include <stdlib.h>

#include "pnl/pnl_random.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_mathtools.h"
#include "carr.h"
#include "levy_process.h"
#include "finance_tool_box.h"
#include "levy_calibration.h"
#include "levy_diffusion_calibration.h"



//double s, double strike_min, double strike_max, int nbr_
    strike, double Tmin, double Tmax, int nbr_maturities, double
    ri, double dividi
int GeneratePrices_ForLevyDiffusion(Calibration_Data *data,
                                    Levy_diffusion *Levy)
{
  int j,k,last;//,error;
  //PnlVect strike_vector,price_vector;
  List_Option_Eqd * op=data->list_model;
  Option_Eqd  op_ref=list_option_eqd_get_value(op,0,0);
  for(j=0;j<op->nb_maturity;j++)
    {
      k=pnl_vect_int_get(op->index_maturity,j);
      last=(j<op->nb_maturity-1)?pnl_vect_int_get(op->ind
    ex_maturity,j+1):op->nb_options;
      /*
      //>> Should be the method used
      //>> but need to work more for out-money options
      //>> need FFT in other function.
      strike_vector=pnl_vect_wrap_subvect(op->K,k,last-k);
      price_vector=pnl_vect_wrap_subvect(op->price,k,last-
    k);

      error=CarrMethod_onStrikeList(&strike_vector,
```

```
                                    &price_vector,
                                    op->S0,
                                    GET(op->T,j),
                                    abs(pnl_vect_int_get(
    op->product,k)-2),
                                    data->rate,
                                    data->divid,
                                    0.01,
                                    Levy);
      */
      //>> For test, consider slowly algorithm :
      op_ref.T=GET(op->T,j);
      while(k<last)
        {
          op_ref.K=GET(op->K,k);
          CarrMethod_Vanilla_option_LD(&op_ref,0.01,Levy);
          LET(op->price,k)=op_ref.price;
          k++;
        }

    }
  return 1;
}


// ----------------------------- Bates_diffusion  ---------
    --------------

void HestonConstraints(PnlVect *res, const PnlVect *x, voi
    d *params)
{

}

double QuadraticError_ForHeston(const PnlVect *Generation
    Params,
                                void *data)
{
  double jump_drift;
  int error;
  Heston_diffusion *Process= Heston_diffusion_create(GET(
```

```
    GenerationParams,2),
                                                    GET(
    GenerationParams,1),
                                                    GET(
    GenerationParams,4),
                                                    GET(
    GenerationParams,3),
                                                    GET(
    GenerationParams,0),&jump_drift);
  Levy_diffusion * Levy =Levy_diffusion_create(Process,&
    Heston_diffusion_characteristic_exponent,&Heston_diffusion_ln_cha
    racteristic_function);
  error= GeneratePrices_ForLevyDiffusion((Calibration_Data*
    )data,Levy);
  free(Levy);
  free(Process);
  return QuadraticError(data);
}

// ----------------------------- Bates_diffusion  ---------
    --------------

void BatesConstraints(PnlVect *res, const PnlVect *x, void
    *params)
{

}




double QuadraticError_ForBates(const PnlVect *Generation
    Params,
                               void *data)
{
  double jump_drift;
  int error;
  Bates_diffusion *Process= Bates_diffusion_create(GET(
    GenerationParams,2),
                                                   GET(
```

```
    GenerationParams,1),
                                                    GET(
    GenerationParams,4),
                                                    GET(
    GenerationParams,3),
                                                    GET(
    GenerationParams,0),
                                                    GET(
    GenerationParams,5),
                                                    GET(
    GenerationParams,6),
                                                    GET(
    GenerationParams,7),
                                                    &jump_dr
    ift);
  Levy_diffusion * Levy =Levy_diffusion_create(Process,&Bat
    es_diffusion_characteristic_exponent,&Bates_diffusion_ln_
    characteristic_function);
  error= GeneratePrices_ForLevyDiffusion((Calibration_Data*
    )data,Levy);
  free(Levy);
  free(Process);
  return QuadraticError(data);
}

// ----------------------------- BNS_diffusion  -----------
    ------------



void BNSConstraints(PnlVect *res, const PnlVect *x, void *
    params)
{
    double sigma0_min, ka_min, eta_min, theta_min, rhow_mi
    n;
    double sigma0_max, ka_max, eta_max, theta_max, rhow_max
    ;

    sigma0_min=0; ka_min=0; eta_min=0; theta_min=0; rhow_mi
    n=-0.9;
    sigma0_max=1; ka_max=5; eta_max=5; theta_max=1; rhow_
```

```
    max=0.9;

    pnl_vect_resize(res, 11);

    LET(res, 0)  = sigma0_max-GET(x, 0);
    LET(res, 1)  = -sigma0_min+GET(x, 0);
    LET(res, 2)  = ka_max-GET(x, 1);
    LET(res, 3)  = -ka_min+GET(x, 1);
    LET(res, 4)  = eta_max-GET(x, 2);
    LET(res, 5)  = -eta_min+GET(x, 2);
    LET(res, 6)  = theta_max-GET(x, 3);
    LET(res, 7)  = -theta_min+GET(x,3 );
    LET(res, 8)  = rhow_max-GET(x, 4);
    LET(res, 9)  = -rhow_min+GET(x, 4);
    LET(res, 10) = 2*GET(x, 1)*GET(x, 2) - GET(x, 3)*GET(x,
     3);
    // Condition de Feller.
}
```

```
double QuadraticError_ForBNS(const PnlVect *GenerationPara
    ms,void *data)
{
  double jump_drift;
  int error;
  BNS_diffusion *Process= BNS_diffusion_create(GET(Generati
    onParams,2),
                                                GET(
    GenerationParams,1),
                                                GET(
    GenerationParams,4),
                                                GET(
    GenerationParams,3),
                                                GET(
    GenerationParams,0),&jump_drift);
  Levy_diffusion * Levy =Levy_diffusion_create(Process,&
    BNS_diffusion_characteristic_exponent,&BNS_diffusion_ln_chara
    cteristic_function);
```

```
  error= GeneratePrices_ForLevyDiffusion((Calibration_Data*
    )data,Levy);
  free(Levy);
  free(Process);
  return QuadraticError(data);
}


// ----------------------------- NIGGammaOU_diffusion  ----
    -------------------

void NIGGammaOUConstraints(PnlVect *res, const PnlVect *x,
    void *params)
{


}




double QuadraticError_ForNIGGammaOU(const PnlVect *Generati
    onParams,void *data)
{
  double jump_drift;
  int error;
  double GammaOU_Alpha=GET(GenerationParams,0);
  double GammaOU_Beta=GET(GenerationParams,1);
  double GammaOU_Delta=GET(GenerationParams,2);
  double OU_Lambda=GET(GenerationParams,3);
  double OU_Alpha=GET(GenerationParams,4);
  double OU_Beta=GET(GenerationParams,5);
  double y0=GET(GenerationParams,6);
  NIG_process *Process= NIG_process_create(GammaOU_Alpha,
    GammaOU_Beta,GammaOU_Delta,&jump_drift);
  GammaOU_diffusion *Time_Clock_Levy=GammaOU_diffusion_crea
    te(OU_Lambda,OU_Alpha,

       OU_Beta,y0,
```

```
      Process, &NIG_process_characteristic_exponent,&jump_dr
    ift);
  Levy_diffusion * Levy =Levy_diffusion_create(Time_Clock_
    Levy,&GammaOU_diffusion_characteristic_exponent,&GammaOU_dif
    fusion_ln_characteristic_function);
  NIG_process_kill_drift(Process);

  error= GeneratePrices_ForLevyDiffusion((Calibration_Data*
    )data,Levy);
  free(Levy);
  free(Process);
  return QuadraticError(data);
}
```

References