```
   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 "levy_diffusion.h"
#include "finance_tool_box.h"
#include "levy_calibration.h"


double TT_interest_rate(double t){return 0.03;};//0.03
// Should be 0 for european and 0.02 for american
double TT_dividend_rate(double t){return 0.00;};//0.03
double TT_volatility(double t){return 0.00;}; //0.15*0.15;}


Calibration_Data * calibration_data_create(List_Option_Eqd
    * list_input_,
                                           double r,
                                           double divid,
                                           int type_of_
    model_)
{

  Calibration_Data  * data = malloc(sizeof(Calibration_Data
    ));
  list_option_eqd_set_rate(list_input_,r,divid);
  data->list_input=list_input_;
  data->list_model=list_option_eqd_copy(list_input_);
  data->type_of_process=type_of_model_;
  return data;
}

void calibration_data_free(Calibration_Data ** data)
{
    if (*data != NULL)
    {
      list_option_eqd_free(&((*data)->list_model));
      free(*data);
      *data=NULL;
    }

}

double calibration_data_QuadraticError(const Calibration_Da
    ta * data)
{
  int i;
  double quad_error=0,diff_price;
  List_Option_Eqd * opmarket=data->list_input;
  List_Option_Eqd * opmodel=data->list_model;
  for(i=0; i<opmarket->nb_options; i++)
    {
      diff_price = GET(opmodel->price,i) - GET(opmarket->
    price,i);
      quad_error += diff_price*diff_price;
    }
  /*
  //>> Error on implied volatility in place of error on
    price
  list_option_eqd_compute_implied_vol(opmodel,data->rate,da
    ta->divid);
  for(i=0; i<opmarket->nb_options; i++)
  {
  diff_price = GET(opmodel->implied_vol,i) - GET(opmarket->
    implied_vol,i);
  quad_error += diff_price*diff_price;
  }
  */
  quad_error = quad_error/opmarket->nb_options;
  // Regularisation term :
  //pnl_vect_print(opmodel->price);
  return quad_error;
}


static int GeneratePrices_ForLevyProcess_fft(Calibration_Da
    ta *data,
                                                  Levy_process *
    Levy)
{
  List_Option_Eqd * op=data->list_model;
  Option_Eqd op_ref;
  int j,k,last;
  PnlVect strike_vector,price_vector;

  //pnl_vect_int_get(op->product,0),op->product_type,op->S0
    ,0,0,0,0);

  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=list_option_eqd_get_value(op,j,k);
          CarrMethod_Vanilla_option(&op_ref,0.01,Levy);
          LET(op->price,k)=op_ref.price;
          //printf(" %7.4f % 7.4f %7.4f {n",opt->T,opt->K,
    LET(op->price,k));
          k++;
        }

    }
  return 1;
}


double QuadraticError_ForLevyProcess_without_cast(Calibrati
    on_Data *data,

    Levy_process *Levy)
{
  int error;
  error= GeneratePrices_ForLevyProcess_fft((Calibration_Da
    ta*)data,Levy);
  return calibration_data_QuadraticError(data);
}


double QuadraticError_ForLevyProcess(const PnlVect *
    GenerationParams,
                                     void *data)
{
  double res;
  Levy_process * Levy =Levy_process_create_from_vect(((Cali
    bration_Data*) data)->type_of_process,GenerationParams->ar
    ray);
  res= QuadraticError_ForLevyProcess_without_cast((Calibr
    ation_Data*)data,Levy);
  Levy_process_free(&Levy);
  return res;

}

void Constraints_ForLevyProcess(PnlVect *res, const PnlVec
    t *x, void *data)
```

```
{
  Levy_process * Levy =Levy_process_create_from_vect(((Cali
    bration_Data*) data)->type_of_process,x->array);
  Levy_process_constraints(res,Levy);
  Levy_process_free(&Levy);
}

static int GeneratePrices_ForLevyDiffusion_fft(Calibration_
    Data *data,

    Levy_diffusion *Levy)
{
  List_Option_Eqd * op=data->list_model;
  Option_Eqd op_ref;
  int j,k,last;
  PnlVect strike_vector,price_vector;

  //pnl_vect_int_get(op->product,0),op->product_type,op->S0
    ,0,0,0,0);

  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;
      strike_vector=pnl_vect_wrap_subvect(op->K,k,last-k);
      price_vector=pnl_vect_wrap_subvect(op->price,k,last-
    k);
      op_ref.T=GET(op->T,j);
      while(k<last)
        {
          op_ref=list_option_eqd_get_value(op,j,k);
          CarrMethod_Vanilla_option_LD(&op_ref,0.01,Levy);
          LET(op->price,k)=op_ref.price;
          //printf(" %7.4f % 7.4f %7.4f {n",op_ref.T,op_ref
    .K,op_ref.price);
          k++;
        }
    }
  return 1;
}
```

```
double QuadraticError_ForLevyDiffusion_without_cast(Calibr
    ation_Data *data,

    Levy_diffusion *Levy)
{
  int error;
  error= GeneratePrices_ForLevyDiffusion_fft((Calibration_
    Data*)data,Levy);
  return calibration_data_QuadraticError(data);
}


double QuadraticError_ForLevyDiffusion(const PnlVect *
    GenerationParams,void *data)
{
  double res;
  Levy_diffusion * Levy =Levy_diffusion_create_from_vect(((
    Calibration_Data*) data)->type_of_process,GenerationParams->
    array);
  res= QuadraticError_ForLevyDiffusion_without_cast((Calibr
    ation_Data*)data,Levy);
  Levy_diffusion_free(&Levy);
  return res;

}

void Constraints_ForLevyDiffusion(PnlVect *res, const PnlV
    ect *x, void *data)
{
  Levy_diffusion * Levy =Levy_diffusion_create_from_vect(((
    Calibration_Data*) data)->type_of_process,x->array);
  Levy_diffusion_constraints(res,Levy);
  Levy_diffusion_free(&Levy);
}
```

References