Help

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

#include "new_cop.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_mathtools.h"
#include "cdo_math.h"
#include "nig.h"




/*
 * Structures to store the paramaters of different copulas
 */
typedef struct {
  double      x1;
  double      x2;
  double      cdf_x1;
  double      cdf_x2;
} element_cdf;

typedef struct {
  double              rho;
  double              g_rho;    /* sqrt(1-rho*rho) */
} gaussian_cop_params;

typedef struct {
  double      theta;
  double      gamma_inv_theta;
  double      pow_theta;
} clayton_cop_params;

typedef struct {
  double      alpha;
  double      beta;
  double      gamma;
  double      mu;
  int         size;
  element_cdf     *data;
} t_nig_cdf;

typedef struct {
  double      rho;
  double      g_rho; /* sqrt(1. - rho*rho); */
  double      alpha;
  double      beta;
  double      gamma;/*sqrt(alphaš-betaš)*/
  double      mu; /* - beta * alpha / gamma; */
  int         size;
  t_nig_cdf       *xcdf;
  t_nig_cdf       *icdf;
} nig_cop_params;

typedef struct {
  double              rho;
  double              g_rho;  /* sqrt(1-rho*rho) */
  double              u_rho;  /* rho/sqrt(1-rho*rho) */
  double              t1;

} student_cop_params;

typedef struct {
  double              rho;
  double              g_rho;  /* sqrt(1-rho*rho) */
  double              u_rho;/* rho/sqrt(1-rho*rho) */
  double              t1;
  double              t2;
  double*             tab1 ;
  double*             tab2 ;

} double_t_cop_params;

static double        GL5_pt[] = { -0.90617984593866399279,
                                 -0.53846931010568309103,
                                 0.,
                                 0.53846931010568309103,
                                 0.90617984593866399279 };
static double        GL5_wg[] = { 0.23692688505618908751,
                                 0.47862867049936646804,
                                 0.5688888888888888888,
                                 0.4786286704993664680 4,
                                  0.2369268850561890875 1 };

////////////////////////////////////////////////////////////
    ////
/* Gaussian copula*/
////////////////////////////////////////////////////////////
    ////

/*
 * Computes the conditionnal probability knowing V
 * f_t = 1 - exp(-intensity * t) for the constant
    intensity case
 */
static double gaussian_compute_prob(const PnlCopula *cop,
    double f_t, double V)
{
  double                  pr;
  gaussian_cop_params    *par;
  double                  p, q, x, mean, sd, bound;
  int                     which,status;

  par = cop->params;
  which=2;
  p=f_t; /* f_t = 1 - exp(-intensity * t) */
  q=1-p;
  mean=0;
  sd=1;
  if (p==0)
    x=-10;
  else
    {
      pnl_cdf_nor(&which, &p, &q, &x, &mean, &sd, &status,
    &bound);
      if (status!=0) { printf("error in pnl_cdf_nor"); abor
    t (); }
    }
  pr = cdf_nor( (x-par->rho*V) / par->g_rho ); //ici beso
    in de V (sachant V)
  return (pr);
```

```
}

static double gaussian_density (const PnlCopula *cop,
    double x){
  return pnl_normal_density(x);
}

PnlCopula* pnl_copula_gaussian_create (double rho)
{
  PnlCopula         *cop;
  gaussian_cop_params  *p;

  p = malloc (sizeof (gaussian_cop_params) );
  p->rho = rho;
  p-> g_rho = sqrt(1-rho*rho);

  cop = malloc (sizeof (PnlCopula) );
  cop->name="Gauss";
  cop->bounds[0] = -6;   cop->bounds[1] = 6;
  cop->proba_cond = gaussian_compute_prob;
  cop->density = gaussian_density;
  cop->params = p;

  return cop;
}


///////////////////////////////////////////////////////////
    ////
/* Clayton copula*/
///////////////////////////////////////////////////////////
    ////


static double clayton_compute_prob(const PnlCopula *cop,
    double f_t, double V)
{
  clayton_cop_params    *par;

  par = cop->params;
  return exp (V * (1. - pow(f_t, -par->theta)));
```

```
}


static double clayton_density(const PnlCopula *cop, double
     x)
{
  clayton_cop_params  *p = cop->params;
  if (x <= 0) return ( 0. );
  return ( (1. / p->gamma_inv_theta) * exp(-x) * pow(x, p->
    pow_theta) );
}

PnlCopula* pnl_copula_clayton_create (double theta)
{
  PnlCopula         *cop;
  clayton_cop_params  *p;

  p = malloc (sizeof (clayton_cop_params) );
  p->theta = theta;
  p->gamma_inv_theta = tgamma(1.0 / theta);
  p->pow_theta = (1.-theta)/theta;

  cop = malloc (sizeof (PnlCopula) );
  cop->name="Clayton";
  cop->bounds[0] = MINDOUBLE;   cop->bounds[1] = 20;
  cop->proba_cond = clayton_compute_prob;
  cop->density = clayton_density;
  cop->params = p;

  return cop;
}


///////////////////////////////////////////////////////////
    ////
/* NIG copula*/
///////////////////////////////////////////////////////////
    ////


static double nig_density(const PnlCopula *cop,double x)
```

```
{
  nig_cop_params *par = cop->params;
  return (nig_generic_density(x, par->alpha, par->beta,
    par->gamma, par->mu, par->alpha));
}


static double nig_cdf(const t_nig_cdf *cdf, double x)
{
  double      min_x;
  double      max_x;
  double      cdf_x;
  double      x0;
  double      s;
  int         i;

  min_x = cdf->data[0].x2;
  max_x = cdf->data[cdf->size-1].x1;
  if ( (x < min_x)||(x > max_x) )  {
    return ( nig_generic_cdf(x, cdf->alpha, cdf->beta, cdf-
    >gamma, cdf->mu, cdf->alpha) );
  }
  else {
    i = (int) ceil((x - min_x) / (max_x - min_x) * (cdf->si
    ze - 1));
    i = (x < cdf->data[i].x1) ? (i-1) : i;
    i = (x > cdf->data[i].x2) ? (i+1) : i;
    cdf_x = cdf->data[i].cdf_x1;
    x0 = cdf->data[i].x1;
    s = 0;
    for (i = 0; i < 5; i++)
      s += GL5_wg[i] * nig_generic_density(x0 + 0.5 * (x -
    x0) * (GL5_pt[i] + 1), cdf->alpha, cdf->beta, cdf->gamma,
    cdf->mu, cdf->alpha);

    return( cdf_x + 0.5 * (x - x0) * s);
  }
}

static void     init_data_cdf(t_nig_cdf     *cdf){
  double      mean = cdf->mu + cdf->alpha * (cdf->beta /
    cdf->gamma);
  double      std_dev = sqrt(cdf->alpha * cdf->alpha * cdf-
    >alpha / (cdf->gamma * cdf->gamma * cdf->gamma));
  double      x;
  double      h;
  double      cdf_x;
  double      s;
  int         i;
  int         j;

  x = mean - 8 * std_dev;
  h = (16. * std_dev) / (double) (cdf->size - 1);
  cdf->data = malloc(cdf->size * sizeof(element_cdf));
  cdf->data[0].x1 = - MAXDOUBLE;
  cdf->data[0].cdf_x1 = 0.;
  cdf->data[0].x2 = x;
  cdf_x = nig_generic_cdf(x, cdf->alpha, cdf->beta, cdf->
    gamma, cdf->mu, cdf->alpha);
  cdf->data[0].cdf_x2 = cdf_x;
  for (i = 1; i < cdf->size-1; i++) {
    cdf->data[i].x1 = cdf->data[i-1].x2;
    cdf->data[i].cdf_x1 = cdf->data[i-1].cdf_x2;
    if (i % 200 == 0) {
      cdf_x = nig_generic_cdf(x+h, cdf->alpha, cdf->beta,
    cdf->gamma, cdf->mu, cdf->alpha);
    }
    else {
      s = 0;
      for (j = 0; j < 5; j++)
        s += GL5_wg[j] * nig_generic_density(x + 0.5 * h *
    (GL5_pt[j] + 1), cdf->alpha, cdf->beta, cdf->gamma, cdf->
    mu, cdf->alpha);
      cdf_x += 0.5 * h * s;
    }
    x += h;
    cdf->data[i].x2 = x;
    cdf->data[i].cdf_x2 = cdf_x;
  }
  cdf->data[i].x1 = cdf->data[i-1].x2;
  cdf->data[i].cdf_x1 = cdf->data[i-1].cdf_x2;
  x += h;
```

```
  cdf->data[i].x2 = MAXDOUBLE;
  cdf->data[i].cdf_x2 = 1.;
}



static int      compare_cdf(const void      *a,
                            const void      *b)
{
  element_cdf     *ea = (element_cdf *) a;
  element_cdf     *eb = (element_cdf *) b;

  if (ea->cdf_x1 < eb->cdf_x1) return (-1);
  if (ea->cdf_x1 > eb->cdf_x2) return (1);
  return (0);
}


static double nig_inv_cdf(const  t_nig_cdf *cdf, double x)
{
   element_cdf     a;
   element_cdf     *r;


  a.cdf_x1 = x;
  r = bsearch(&a, cdf->data, cdf->size, sizeof(element_cdf)
    , compare_cdf);
  if (r->cdf_x1 == 0)
    return ( r->x2 + log(x / r->cdf_x2) );
  if (r->cdf_x2 == 1)
    return ( r->x1 - log((1 - x) / (1 - r->cdf_x1)) );
  return ( r->x1 + (r->x2 - r->x1)/(r->cdf_x2 - r->cdf_x1)
    * (x - r->cdf_x1) );
}

static double nig_compute_prob(const PnlCopula *cop,
    double f_t, double V)
{
  double x;
  nig_cop_params    *par;
  par = cop->params;
  x=(nig_inv_cdf(par->icdf,f_t)-par->rho*V)/par->g_rho;
  return nig_cdf(par->xcdf,x);
}


PnlCopula* pnl_copula_nig_create (double rho, double alpha,
     double beta)
{
  PnlCopula         *cop;
  nig_cop_params  *p;
  cop = malloc (sizeof (PnlCopula));
  cop->name="NIG";
  p = malloc (sizeof (nig_cop_params) );
  p->rho = rho;
  p->g_rho = sqrt(1. - rho*rho);
  p->alpha = alpha;
  p->beta = beta;
  p->gamma=sqrt(alpha * alpha - beta * beta);
  p->mu=-alpha*beta/p->gamma;

  p->icdf = malloc(sizeof(t_nig_cdf));
  p->icdf->alpha = alpha / rho;
  p->icdf->beta = beta / rho;
  p->icdf->gamma = p->gamma / rho;
  p->icdf->mu = p->mu / rho;
  p->icdf->size = 10000;
  init_data_cdf(p->icdf);


  p->xcdf = malloc(sizeof(t_nig_cdf));
  p->xcdf->alpha = alpha * p->g_rho / rho;
  p->xcdf->beta = beta * p->g_rho / rho;
  p->xcdf->gamma = p->gamma * p->g_rho / rho;
  p->xcdf->mu = p->mu * p->g_rho / rho;
  p->xcdf->size = 10000;
  init_data_cdf(p->xcdf);


  cop->bounds[0] = -12;   cop->bounds[1] = 12;
  cop->proba_cond = nig_compute_prob;
  cop->density = nig_density;
  cop->params = p;
  return cop;
}

///////////////////////////////////////////////////////////
    ////
/* Student copula*/
///////////////////////////////////////////////////////////
    ////

static double  student_density(const PnlCopula *cop,
    double x){
  student_cop_params *p;
  p=cop->params;
  return (tgamma((p->t1+1)*0.5)/((tgamma((p->t1)*0.5))*sq
    rt(M_PI*(p->t1))*exp((((p->t1)+1)*0.5)*log(1+x*x/(p->t1)))))
    ;
}

static double student_compute_prob(const PnlCopula *cop,
    double f_t, double V)
{
  student_cop_params *p;
  double a;
  p = cop->params;
  a=student_inv_cdf(p->t1,f_t)/(p->g_rho);
  return cdf_nor(a*sqrt(V/p->t1) - (p->u_rho *V)); //bizarr
    e car Vincent ne met pas le même V
}

PnlCopula* pnl_copula_student_create(double rho, double t1)
{
  PnlCopula               *cop;
  student_cop_params      *p;

  p = malloc(sizeof(student_cop_params));
  p->rho = rho;
  p->g_rho = sqrt(1.0 - rho*rho);
  p->u_rho = rho / p->g_rho;
  p->t1=t1;
  cop = malloc(sizeof(PnlCopula));
  cop->name = "One-factor Student Copula";
  cop->bounds[0] = -12;   cop->bounds[1] = 12;//faux, on
    ne peut pas aller
                                              //jusque -12
    il y a une
                                              //racine. qu
    elles sont les bornes?
  cop->proba_cond = student_compute_prob;
  cop->density = student_density;
  cop->params = p;
  return (cop);
}



///////////////////////////////////////////////////////////
    ////
/* Double-t copula*/
///////////////////////////////////////////////////////////
    ////


static double *init_points(PnlCopula *cop){
  int i;
  double a1,b1;
  int n=500;
  double *tab;
  double_t_cop_params *p;
  p=cop->params;
  tab=malloc(n*sizeof(double));

  a1=-6*sqrt((p->t1-2)/p->t1)*(p->rho)-6*sqrt((p->t2-2)/p->
    t2)*sqrt(1-(p->rho)*(p->rho));
  b1=6*sqrt((p->t1-2)/p->t1)*(p->rho)+6*sqrt((p->t2-2)/p->
    t2)*sqrt(1-(p->rho)*(p->rho));

  for(i=0;i<n;i++){
    tab[i]=a1+i*(b1-a1)*1./(n-1);
  }
  return tab;
```

```
}

static double f1(double rho,double t1,double x1){
  double u=exp((t1+1)*0.5*log(1+x1*x1/(rho*rho*(t1-2))));
  return 1/(u);
}

static double f2(double rho,double t2,double x2){
  double u=exp((t2+1)*0.5*log(1+x2*x2/((1-rho*rho)*(t2-2)))
    );
  return 1/(u);
}




static double *init_cdf(PnlCopula *cop)
{

  double_t_cop_params *p;
  int l;
  int n=0;
  double fval1,h1,x1;
  double a1=0;
  double b1=0;
  double a2=0;
  double pi=3.14159265;
  double s1=0;
  double *s2;
  double k;
  int i,j;
  double coefs=0;

  p=cop->params;
  n=500;
  a1=-6*(p->rho)*sqrt((p->t1-2)/p->t1);
  b1=6*(p->rho)*sqrt((p->t1-2)/p->t1);
  a2=-6*sqrt(1-(p->rho)*(p->rho))*sqrt((p->t2-2)/p->t2);
  s2=malloc(n*sizeof(double));
  coefs=tgamma((p->t1+1)*0.5)*tgamma((p->t2+1)*0.5)/((tgam
    ma(p->t1*0.5))*(tgamma(p->t2*0.5))*pi*(p->rho)*sqrt((1-p->rh
    o*p->rho)*(p->t1-2)*(p->t2-2)));
  k=(b1-a1)*1./n;

  for(l=0;l<n;l++){
    s2[l]=0;
    for(i=0;i<n;i++){
      s1=0;
      x1=p->tab1[l]-(a1+k*i);
      if(x1>a2){
        h1=(x1-a2)*1./n;
        for(j=0;j<n;j++){

          fval1=f2(p->rho,p->t2,a2+h1*j);

          s1+=fval1;
        }
        s1=s1*h1;
        s2[l]+=s1*coefs*k*f1(p->rho,p->t1,k*i+a1);

      }
    }
  }

  return s2;
}
```

```
static double double_t_inv_cdf(const PnlCopula *cop,
    double x)
{
  int n=0;
  int i,u,v;
  double_t_cop_params *p;
  int a;

  n=500;
  p=cop->params;
  a=0;

  if(x==1) x=1-0.0001;

  if(x==0){
    do{
      a=a+1;
    }while(p->tab2[a]==0);

    return p->tab1[a];
  }
  u=0;
  v=n;
  i=1;
  a=0;

  if ((x<0)||(x>1)) return 0;

  if(p->tab2[0]>=x) return p->tab1[0];
  else if (x>p->tab2[n-1]) return p->tab1[n-1];


  do{
    i=i+1;
  }while(p->tab2[i]<x);
  a=i-1;

  return p->tab1[a] +((x-p->tab2[a])*(p->tab1[a+1]-p->tab1[
    a]))/((p->tab2[a+1]-p->tab2[a]));
}

static double  double_t_density(const PnlCopula *cop,
    double x){
  double_t_cop_params *p;
  p=cop->params;
  return (tgamma((p->t1+1)*0.5)/((tgamma((p->t1)*0.5))*sq
    rt(M_PI*(p->t1))*exp((((p->t1)+1)*0.5)*log(1+x*x/(p->t1)))))
    ;
}
```

```
static double double_t_compute_prob(const PnlCopula *cop,
    double f_t, double V)
{
  double_t_cop_params     *p;
  double              a;
  double              b;
  p = cop->params;

  a=double_t_inv_cdf(cop,f_t);
  b=sqrt(p->t2/(p->t2-2))*1./p->g_rho;

  return student_cdf(p->t2,b*(a-p->rho*sqrt((p->t1-2)/(p->
    t1))*V));
}



PnlCopula* pnl_copula_double_t_create(double rho, double t1
    , double t2)
{
  PnlCopula              *cop;
  double_t_cop_params      *p;

  cop = malloc(sizeof(PnlCopula));
  cop->name = "One-factor Double-t Copula";
  p = malloc(sizeof(double_t_cop_params));
  cop->params = p;
  p->rho = rho;
  p->g_rho = sqrt(1.0 - rho*rho);
  p->u_rho = rho / p->g_rho;
  p->t1=t1;
  p->t2=t2;
  p->tab1=init_points(cop);
  p->tab2=init_cdf(cop);
  cop->bounds[0] = -6;   cop->bounds[1] = 6;//faux, on ne
    peut pas aller
                                                //jusque -12
    il y a une
                                                //racine. qu
    elles sont les bornes?
  cop->proba_cond = double_t_compute_prob;
  cop->density = double_t_density;

  return (cop);
}

///////////////////////////////////////////////////////////
    /////
/* Copula initialization*/
///////////////////////////////////////////////

PnlCopula *pnl_copula_create (int t_copula, const double *
    p_copula)
{
  PnlCopula *cop;
  switch (t_copula) {
  case 1 :
    cop = pnl_copula_gaussian_create (p_copula[0]);
    break;
  case 2 :
    cop = pnl_copula_clayton_create(p_copula[0]);
    break;
  case 3 :
    cop = pnl_copula_nig_create(p_copula[0], p_copula[1],
    p_copula[2]);
    break;
    /* case 4:
     *   cop= pnl_copula_student_create( p_copula[0],p_    copula[1]);
     *   break; */
  case 5:
    cop=  pnl_copula_double_t_create( p_copula[0],p_copula[
    1],p_copula[2]);
    break;
  default:
    return NULL;
  }
  return cop;
}

void pnl_copula_free (PnlCopula **cop)
{
  free ((*cop)->params);
  free (*cop); *cop = NULL;
}
```

References