Help

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

#include "
href../../../common/math/cdo/copulas_h_src.pdfcopulas.h"
#include "pnl/pnl_random.h"

/** Parameters of the Clayton copula.
 */
typedef struct
{
  double      theta;
  double      gamma_inv_theta;
  double      pow_theta;
  double      factor;
} clayton_params;


/* Computes the density of the Gamma distribution with parameter \ f$\ frac{1}{\
 * \ f[
 *  f(x) = \ frac{1}{\ Gamma(\ frac{1}{\ theta})} \ exp(-x) x^{(1-\ theta)/\ the
 * \ f]
 * Used in the Hull&White and Laurent&Gregory approaches.
 */
static double       clayton_density(const copula        *cop,
                                    const double        x)
{
  clayton_params  *p = cop->parameters;
  return ((1. / p->gamma_inv_theta) * exp(-x) * pow(x, p->pow_theta));
}


/* Computes the conditional default probabilities \ f$p_t^{i|V}\ f$.
 * \ param cop the clayton copula
 * \ param f_t a double containing \ f$F_i(t)\ f$ the cdf of default time \ f$\
 * \ param v a pointer on a grid discrtizing the factor V
 * \ return an array of double of size \ c cop->size
 * \ note Used in the Hull&White and Laurent&Gregory approaches, not in Monte-Ca
```

```
 */
static double       *clayton_compute_prob(const copula      *cop,
    const double      f_t)
{
  double         *result;
  clayton_params *p      = cop->parameters;
  int             i;

  result = malloc(cop->size * sizeof(double));
  for (i = 0; i < cop->size; i++)
    {
      result[i] = exp(cop->points[i] * (1. - pow(f_t, -p->theta)));
    }
  return (result);
}


/* Generates a Gamma distributed random variable by a reject method.
 * \ note Used in the Monte-Carlo approach, not in Hull&White and Laurent&Gregor
 */
static void     gamma_generate(copula           *cop)
{
  clayton_params  *p = cop->parameters;
  double      a = 1 / p->theta;
  double      a_ = a - 1;
  double      b = (a - (1 / (6 * a))) / a_;
  double      m = 2 / a_;
  double      d = m + 2;
  double      U1;
  double      U2;
  double      V;
  int         accept = 0;

  do
    {
      U1 = pnl_rand_uni(0);
      U2 = pnl_rand_uni(0);
      V = b * U2 / U1;
      if (m * U1 - d + V + (1 / V) <= 0) accept++;
      else if (m * log(U1) - log(V) + V - 1 <= 0) accept++;
    }
```

```
  while (accept == 0);

  p->factor = a_ * V;
}


/* Computes the default time \ f$\ tau_i\ f$ defined by
 * \ f[
 *  \ tau_i = F_i^{-1}(\ Psi(- \ log(U_i) / V))
 * \ f]
 * where \ f$\ Psi(s)=(1+s)^{-1/\ theta}\ f$ and \ f$F_i^{-1}\ f$ the generalize
 * \ warning Works only if the intensity \ f$h_i\ f$ is constant !
 * \ note Used in the Monte-Carlo approach, not in Hull&White and Laurent&Gregor
 */
static int      clayton_compute_dt(const copula     *cop,
                                   const step_fun   *H,
                                   double       *time)
{
  clayton_params  *p = cop->parameters;
  double      U_i;
  double      cdf_V_i;
  double      zi;

  U_i = pnl_rand_uni(0);
  cdf_V_i = pow(1. - log(U_i) / p->factor, -1 / p->theta);
  zi = -log(1. - cdf_V_i);
  if (zi >= H->data[H->size - 1].y2) return (0);
  else
    {
      *time = inverse_sf(H, zi);
      return (1);
    }
}

/* Initialization of the one-factor Clayton Copula.
 */
copula          *init_clayton_copula(const double   theta)
{
  copula      *cop;
  clayton_params  *p;
  double      h;
```

```
  double      v0;
  int         jv;

  cop = malloc(sizeof(copula));
  cop->name = "One-factor Clayton Copula";
  cop->nfactor = 1;
  p = malloc(sizeof(clayton_params));
  p->theta = theta;
  p->gamma_inv_theta = pnl_tgamma(1.0 / theta);
  p->pow_theta = (1. - theta) / theta;
  cop->parameters = p;
  cop->size = 200;
  cop->points = malloc(cop->size * sizeof(double));
  cop->weights = malloc(cop->size * sizeof(double));
  h = 20. / (cop->size - 1);
  for (jv = 0, v0 = MINDOUBLE; jv < cop->size; jv++, v0 += h)
    {
      cop->points[jv] = v0;
      cop->weights[jv] = clayton_density(cop, v0) * h;
    }
  cop->density = clayton_density;
  cop->compute_cond_prob = clayton_compute_prob;
  cop->generate = gamma_generate;
  cop->compute_default_time = clayton_compute_dt;

  return (cop);
}
```