

[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"
#include "pnl/pnl_specfun.h"

typedef struct
{
    double      x1;
    double      x2;
    double      cdf_x1;
    double      cdf_x2;
} element_cdf;

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

typedef struct
{
    double      a;
    double      g_a;
    double      alpha;
    double      beta;
    double      gamma;
    double      mu;
    double      factor;

    step_fun    *cdf_Xi;
    step_fun    *cdf_Ai;
```

```

    t_nig_cdf      *xcdf;
    t_nig_cdf      *icdf;
} nig_params;

const double      GL5_pt[] = { -0.90617984593866399279,
                                -0.53846931010568309103,
                                0.,
                                0.53846931010568309103,
                                0.90617984593866399279
                                };
const double      GL5_wg[] = { 0.23692688505618908751,
                                0.47862867049936646804,
                                0.5688888888888888888888,
                                0.47862867049936646804,
                                0.23692688505618908751
                                };

static double      nig_generic_density(const double      x,
                                       const double      alpha,
                                       const double      beta,
                                       const double      gamma,
                                       const double      mu,
                                       const double      delta)
{
    double          f_x = sqrt(delta * delta + (x - mu) * (x - mu));
    return ((delta * alpha * exp(delta * gamma + beta * (x - mu)) *
            pnl_bessel_k(1, alpha * f_x)) / (M_PI * f_x));
}

static double      ig_generic_density(const double      y,
                                       const double      alpha,
                                       const double      beta)
{
    double          z = alpha - beta * y;

    if (y <= 0) return (0.);
    return (M_1_SQRT2PI * (alpha / sqrt(beta)) * pow(y, -1.5) * exp(- z * z / (2.
}

static double      nig_generic_cdf(const double      x,
                                   const double      alpha,

```

```

                                const double      beta,
                                const double      gamma,
                                const double      mu,
                                const double      delta)
{
    double      y;
    double      z;
    double      t;
    double      h;
    double      s1;
    double      s2;

    s1 = 0;
    h = 4. / 100.;
    for (y = MINDOUBLE; y < 4.; y += h)
    {
        z = (x - (mu + beta * (y + 0.5 * h))) / sqrt(y + 0.5 * h);
        s1 += cdf_nor(z) * ig_generic_density(y + 0.5 * h, delta * gamma, gamma *
    }
    s1 *= h;
    s2 = 0;
    h = exp(-4.) / 20.;
    for (t = MINDOUBLE; t < exp(-4.); t += h)
    {
        y = -log(t + 0.5 * h);
        z = (x - (mu + beta * y)) / sqrt(y);
        s2 += cdf_nor(z) * ig_generic_density(y, delta * gamma, gamma * gamma) * (
    }
    s2 *= h;

    return (s1 + s2);
};

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
    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->al
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->
    }
    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_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.;

return ;
}

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,
                               const double      cdf_x)
{
    element_cdf    a;
    element_cdf    *r;

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

static double      nig_cdf(const t_nig_cdf    *cdf,
                           const 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
}

```

```

else
{
    i = (int) ceil((x - min_x) / (max_x - min_x) * (cdf->size - 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] +

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

static double      nig_density(const copula      *cop,
                               const double      x)
{
    nig_params      *p = cop->parameters;

    return (nig_generic_density(x, p->alpha, p->beta, p->gamma, p->mu, p->alpha));
}

static double      *nig_compute_prob(const copula      *cop,
                                      const double      f_t)
{
    double          *result;
    nig_params      *p = cop->parameters;
    double          C;
    int             jv;

    result = malloc(cop->size * sizeof(double));
    C = nig_inv_cdf(p->icdf, f_t);
    for (jv = 0; jv < cop->size; jv++)
    {
        result[jv] = nig_cdf(p->xcdf, (C - p->a * cop->points[jv]) / p->g_a);
    }
}

```

```

    return (result);
}

```

```

static double      nig_generic_generate(const double  alpha,
    const double    beta,
    const double    gamma,
    const double    mu,
    const double    delta)
{
    double  chi = pow(pnl_rand_normal(0), 2.);
    double  tau = delta / gamma;
    double  lambda = delta * delta;
    double  z;

    z = tau + tau * (tau * chi - sqrt(tau * chi * (4 * lambda + tau * chi))) / (2 * tau);
    z = (pnl_rand_uni(0) <= tau / (tau + z)) ? z : (tau * tau / z);

    return (mu + beta * z + sqrt(z) * pnl_rand_normal(0));
}

```

```

static void      nig_generate(copula      *cop)
{
    nig_params  *p = cop->parameters;

    p->factor = nig_generic_generate(p->alpha, p->beta, p->gamma, p->mu, p->alpha);
}

```

```

static int      nig_compute_dt(const copula      *cop,
    const step_fun      *H,
    double            *time)
{
    nig_params      *p = cop->parameters;
    double          Vi;
    double          zi;

    p = cop->parameters;
    Vi = p->a * p->factor + p->g_a * nig_generic_generate(p->xcdf->alpha, p->xcdf->beta, p->xcdf->gamma, p->xcdf->mu, p->xcdf->alpha);
}

```

```

zi = -log(1. - nig_cdf(p->icdf, Vi));
if (zi >= H->data[H->size - 1].y2) return (0);
else
{
    *time = inverse_sf(H, zi);
    return (1);
}
}

```

```

copula          *init_nig_copula(const double      a,
                                const double      alpha,
                                const double      beta)
{
    copula      *cop;
    nig_params  *p;
    double      v0;
    double      h;
    int         jv;

    cop = malloc(sizeof(copula));
    cop->name = "One-factor NIG Copula";
    cop->nfactor = 2;
    p = malloc(sizeof(nig_params));
    p->a = a;
    p->g_a = sqrt(1. - a * a);
    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 / a;
    p->icdf->beta = beta / a;
    p->icdf->gamma = p->gamma / a;
    p->icdf->mu = p->mu / a;
    p->icdf->size = 10000;
    init_data_cdf(p->icdf);

    p->xcdf = malloc(sizeof(t_nig_cdf));

```



```

p->xcdf->alpha = alpha * p->g_a / a;
p->xcdf->beta = beta * p->g_a / a;
p->xcdf->gamma = p->gamma * p->g_a / a;
p->xcdf->mu = p->mu * p->g_a / a;
p->xcdf->size = 10000;
init_data_cdf(p->xcdf);

cop->parameters = p;
cop->size = 200;
cop->points = malloc(cop->size * sizeof(double));
cop->weights = malloc(cop->size * sizeof(double));
h = 24. / (cop->size - 1);
for (jv = 0, v0 = -12.; jv < cop->size; jv++, v0 += h)
{
    cop->points[jv] = v0;
    cop->weights[jv] = nig_density(cop, v0) * h;
}
cop->density = nig_density;
cop->compute_cond_prob = nig_compute_prob;
cop->generate = nig_generate;
cop->compute_default_time = nig_compute_dt;

return (cop);
}

```