

[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"

typedef struct
{
    double          rho;
    double          g_rho;
    double          u_rho;
    double          factor1;
    double          factor2;
    double          t1;

} Student_params;

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

double simulate_student(double t)
{
    double s = 0;
    int j;
    double u;
    for (j = 0; j < t; j++)
    {
        u = pnl_rand_normal(PNL_RNG_KNUTH);
        s += u * u;
    }
    return pnl_rand_normal(PNL_RNG_KNUTH) * sqrt(t / s);
}

static double simulate(double t)
{
    double s = 0;
```

```

    int j;
    double u;
    for (j = 0; j < t; j++)
    {
        u = pnl_rand_normal(PNL_RNG_KNUTH);
        s += u * u;
    }
    return t / s;
}

double student_cdf(double t, double x)
{
    int which = 1, status;
    double bound, p, q;
    pnl_cdf_t (&which, &p, &q, &x, &t, &status, &bound);
    if (status != 0)
    {
        printf("error in pnl_cdf_t");
        abort();
    }
    return p;
}

double student_inv_cdf(double t, double p)
{
    int which = 2, status;
    double bound, q, x;
    if (p <= 0.) p = 0.001;
    if (p >= 1.) p = 0.999;
    q = 1. - p;
    pnl_cdf_t (&which, &p, &q, &x, &t, &status, &bound);
    if (status != 0)
    {
        printf("error in pnl_cdf_t");
        abort();
    }
    return x;
}

static void Student_generate(copula *cop)
{

```

```

Student_params *p;
p = cop->parameters;
((Student_params *) cop->parameters)->factor1 = pnl_rand_normal(0);
((Student_params *) cop->parameters)->factor2 = sqrt(simulate(p->t1));
}

static double Student_density(const copula *cop, const double x)
{
    Student_params *p;
    p = cop->parameters;
    return (pnl_tgamma((p->t1 + 1) * 0.5) / ((pnl_tgamma((p->t1) * 0.5)) * sqrt(M_
        * exp((((p->t1) + 1) * 0.5) * log(1 + x * x / (p->t1))))));
}

static double *Student_compute_prob(const copula *cop, const double f_t)
{
    double *result;
    Student_params *p;
    double a;
    int i, j;
    int h = (cop->size) * (cop->size);
    p = cop->parameters;
    result = malloc(h * sizeof(double));

    a = student_inv_cdf(p->t1, f_t) / (p->g_rho);

    for (i = 0; i < cop->size; i++)
    {
        for (j = 0; j < cop->size; j++)
        {
            result[j + i * cop->size] = cdf_nor(a * sqrt(cop->points[i + cop->size
                - (p->u_rho * cop->points[j])));
        }
    }
    return (result);
}

static int Student_compute_dt(const copula *cop, const step_fun *H, double *tim
{
    Student_params *p;
    double X;

```

```

double zi;
p = cop->parameters;
X = (p->rho * p->factor1 + p->g_rho * pnl_rand_normal(0)) * p->factor2;
zi = -log(1. - student_cdf(p->t1, X));
if (zi >= H->data[H->size - 1].y2) return (0);
else
{
    *time = inverse_sf(H, zi);
    return (1);
}
}

static double density_chi2(double t1, double x)
{
    double a = 1 / (exp((t1 * 0.5) * log(2.)) * pnl_tgamma(t1 * 0.5));
    if (x > 0)
    {
        return a * exp((t1 * 0.5 - 1) * log(x)) * exp(-x * 0.5);
    }
    else return 0;
}

copula *init_student_copula(const double rho, const double t1)
{
    /* Gauss quadrature weights and kronrod quadrature abscissae and
       weights as evaluated with 80 decimal digit arithmetic by
       L. W. Fullerton, Bell Labs, Nov. 1981. */

    /* abscissae of the 21-point kronrod rule */
    double xgk[12] =
    {
        0.0,
        0.99565716302580808073552728070,
        0.97390652851717172007796401210,
        0.93015749135570822600120718010,
        0.86506336668898451073209668840,
        0.78081772658641689706371757830,
        0.67940956829902440623432736510,
        0.56275713466860468333900009930,
        0.43339539412924719079926594320,

```

```

    0.29439286270146019813112660310,
    0.14887433898163121088482600110,
    0.0
};

/* weights of the 21-point gauss rule */
double wgk[12] =
{
    0.0,
    0.011694638867371874278064396060,
    0.032558162307964727478818972460,
    0.054755896574351996031381300240,
    0.075039674810919952767043140920,
    0.093125454583697605535065465080,
    0.10938715880229764189921059030,
    0.12349197626206585107795810980,
    0.13470921731147332592805400180,
    0.14277593857706008079709427310,
    0.14773910490133849137484151600,
    0.14944555400291690566493646840
};

copula          *cop;
Student_params  *p;
int             jv;
int             b;
double          a1 = -6.0;
double          b1 = 6.0;
double          a2 = 0.0;
double          b2 = 12.0;

cop = malloc(sizeof(copula));
cop->name = "Two-factor Student Copula";
cop->nfactor = 2;
p = malloc(sizeof(Student_params));
cop->parameters = p;
p->rho = rho;
p->g_rho = sqrt(1.0 - rho * rho);
p->u_rho = rho / p->g_rho;
p->t1 = t1;
cop->size = 22;

```

```

b = 2 * (cop->size);
cop->points = malloc(b * sizeof(double));
cop->weights = malloc(b * sizeof(double));

cop->points[0] = xgk[0];
cop->points[0 + cop->size] = xgk[0];
cop->weights[0] = wgk[0];
cop->weights[0 + cop->size] = wgk[0];
for (jv = 1; jv < 11; jv++)
{
    cop->points[jv] = (a1 + b1) * 0.5 + (b1 - a1) * 0.5 * xgk[jv];
    cop->points[jv + 10] = (a1 + b1) * 0.5 - (b1 - a1) * 0.5 * xgk[jv];
    cop->weights[jv] = wgk[jv] * gaussian_density(cop, cop->points[jv]) * (b1 - a1);
    cop->weights[jv + 10] = wgk[jv] * gaussian_density(cop, cop->points[jv + 10]) * (b1 - a1);

    cop->points[jv + cop->size] = (a2 + b2) * 0.5 + (b2 - a2) * 0.5 * xgk[jv];
    cop->points[jv + 10 + cop->size] = (a2 + b2) * 0.5 - (b2 - a2) * 0.5 * xgk[jv];

    cop->weights[jv + cop->size] = wgk[jv] * density_chi2(p->t1, cop->points[jv]);
    cop->weights[jv + 10 + cop->size] = wgk[jv] * density_chi2(p->t1, cop->points[jv + 10]);
}

cop->points[21] = (a1 + b1) * 0.5;
cop->weights[21] = (b1 - a1) * 0.5 * wgk[11] * gaussian_density(cop, cop->points[21]);

cop->points[43] = (a2 + b2) * 0.5;
cop->weights[43] = (b2 - a2) * 0.5 * wgk[11] * density_chi2(p->t1, cop->points[43]);
cop->density = Student_density;
cop->generate = Student_generate;
cop->compute_default_time = Student_compute_dt;
cop->compute_cond_prob = Student_compute_prob;

return (cop);
}

```