

## [Help](#)

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

#include "
href../../../../mod/copula/copula_stdndc/copula_stdndc_h_src.pdfcopula_stdndc.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "
href../../../../common/math/cdo/cdo_h_src.pdfmath/cdo/cdo.h"
#include "
href../../../../mod/copula/copula_stdndc/price_cdo_h_src.pdfprice_cdo.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_fft.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(LaurentGregory)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(LaurentGregory)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

double          **lg_numdef(const CDO          *cdo,
                           const copula        *cop,
                           const grid          *t,
                           const cond_prob     *cp)
{
    double        **nd;
    double        *r;
    double        *r_cpy;
    int           jt;
    int           jr;
    int           jn;
```

```

int          jv;

nd = malloc(t->size * sizeof(double *));
r = malloc((cdo->n_comp + 1) * sizeof(double));
r_cpy = malloc((cdo->n_comp + 1) * sizeof(double));
for (jt = 0; jt < t->size; jt++)
{
    nd[jt] = malloc((cdo->n_comp + 1) * sizeof(double));
    for (jr = 0; jr < cdo->n_comp + 1; jr++)
    {
        nd[jt][jr] = 0;
    }
    for (jv = 0; jv < cop->size; jv++)
    {
        r[0] = 1.;
        for (jr = 1; jr < cdo->n_comp + 1; jr++)
            r[jr] = 0;
        for (jn = 0; jn < cdo->n_comp; jn++)
        {
            for (jr = 0; jr < jn + 1; jr++)
            {
                r_cpy[jr] = r[jr];
            }
            r[0] = (1. - cp->p[jn][jt][jv]) * r_cpy[0];
            for (jr = 1; jr < jn + 2; jr++)
            {
                r[jr] += cp->p[jn][jt][jv] * (r_cpy[jr - 1] - r[jr]);
            }
        }
        for (jr = 0; jr < cdo->n_comp + 1; jr++)
        {
            nd[jt][jr] += r[jr] * cop->weights[jv];
        }
    }
}

free(r);
free(r_cpy);

return (nd);
}

```

```

double          **lg_losses(const CDO      *cdo,
                           const copula    *cop,
                           const grid      *t,
                           const grid      *x,
                           const cond_prob *cp)
{
    double          **losses;
    grid            *u;
    dcomplex        *phi_cov;
    dcomplex        prod;
    int             jt;
    int             ju;
    int             jv;
    int             jn;
    PnlVectComplex  *fft_in;
    PnlVectComplex  *fft_out;
    double          C;
    double          F;

    fft_in = pnl_vect_complex_create(x->size);
    fft_out = pnl_vect_complex_create(x->size);
    u = create_grid(x->size);
    C = x->data[x->size - 1];
    F = (M_2PI * pow(x->size - 1., 2.)) / (2. * C * (double) x->size);
    for (ju = 0; ju < x->size; ju++)
        u->data[ju] = - F + 2.*F * (ju / ((double) x->size - 1.));
    phi_cov = malloc(cdo->n_comp * sizeof(dcomplex));
    losses = malloc(t->size * sizeof(double *));
    for (jt = 0; jt < t->size; jt++)
    {
        pnl_vect_complex_set_dcomplex(fft_in, CZERO);
        for (ju = 0; ju < x->size; ju++)
        {
            for (jn = 0; jn < cdo->n_comp; jn++)
            {
                phi_cov[jn] = PHI_COV(jn, u->data[ju] * cdo->C[jn]->nominal);
            }
            for (jv = 0; jv < cop->size; jv++)
            {

```

```

        prod = CONE;
        for (jn = 0; jn < cdo->n_comp; jn++)
        {
            prod = Cmul(prod,
                        CRadd(RCmul(cp->p[jn][jt][jv],
                                    CRadd(phi_cov[jn], - 1.)),
                                1.0));
        }
        pnl_vect_complex_set(fft_in, ju,
                            Cadd(pnl_vect_complex_get(fft_in, ju),
                                RCmul(cop->weights[jv], prod)));
    }
}
pnl_fft(fft_in, fft_out);
losses[jt] = malloc(x->size * sizeof(double));
for (ju = 0; ju < x->size; ju++)
{
    losses[jt][ju] = (C / ((double) x->size - 1.)) *
                    Creal(Cmul(CIexp(F * x->data[ju]),
                                pnl_vect_complex_get(fft_out, ju)))
                    * 2.*F / (M_2PI * ((double) x->size - 1.));
}
}
pnl_vect_complex_free(&fft_in);
pnl_vect_complex_free(&fft_out);
free(phi_cov);
free_grid(u);
return (losses);
}

```

```

double          **lg_numdef1(const CDO      *cdo,
                             const copula   *cop,
                             const grid      *t,
                             const cond_prob *cp)
{
    double      **nd;
    double      **r;
    double      **r_cpy;
    double      *z ;

```

```

int          jt;
int          jr;
int          jn;
int          jv, jw;

nd = malloc(t->size * sizeof(double *));

r = malloc(((cdo->n_comp + 1)) * sizeof(double *));
r_cpy = malloc(((cdo->n_comp + 1)) * sizeof(double *));

for (jr = 0; jr < cdo->n_comp + 1; jr++)
{
    r[jr] = malloc((cop->size) * sizeof(double));
    r_cpy[jr] = malloc((cop->size) * sizeof(double));
}
z = malloc((cdo->n_comp + 1) * sizeof(double));

for (jt = 0; jt < t->size; jt++)
{
    nd[jt] = malloc((cdo->n_comp + 1) * sizeof(double));
    for (jr = 0; jr < cdo->n_comp + 1; jr++)
    {
        nd[jt][jr] = 0;
        z[jr] = 0;
    }

    for (jv = 0; jv < cop->size; jv++)
    {
        for (jw = 0; jw < cop->size; jw++)
        {
            r[0][jw] = 1.;
            for (jr = 1; jr < cdo->n_comp + 1; jr++)
            {
                r[jr][jw] = 0;
            }

            for (jn = 0; jn < cdo->n_comp; jn++)
            {
                for (jr = 0; jr < jn + 1; jr++)
                {

```

```

        r_cpy[jr][jw] = r[jr][jw];
    }
    r[0][jw] = (1. - cp->p[jn][jt][jv + jw * cop->size]) * r_cpy[0]
    for (jr = 1; jr < jn + 2; jr++)
    {
        r[jr][jw] += cp->p[jn][jt][jv + jw * cop->size] * (r_cpy[jr]
    }
    }
    for (jr = 0; jr < cdo->n_comp + 1; jr++)
    {
        z[jr] += r[jr][jw] * (cop->weights[jw + cop->size]);
    }
}

for (jr = 0; jr < cdo->n_comp + 1; jr++)
{
    nd[jt][jr] += z[jr] * cop->weights[jv];
    z[jr] = 0;
}
}

}
free(r);
return (nd);
}

double          **lg_losses1(const CDO      *cdo,
                             const copula   *cop,
                             const grid     *t,
                             const grid     *x,
                             const cond_prob *cp)
{
    double          **losses;
    grid            *u;
    dcomplex        *phi_cov;
    dcomplex        prod;
    dcomplex        prod1;
    int             jt;
    int             ju;
    int             jv;
    int             iv;
    int             jn;

```

```

PnlVectComplex    *fft_in;
PnlVectComplex    *fft_out;
double            C;
double            F;

fft_in = pnl_vect_complex_create(x->size);
fft_out = pnl_vect_complex_create(x->size);
u = create_grid(x->size);
C = x->data[x->size - 1];
F = (M_2PI * pow(x->size - 1., 2.)) / (2. * C * (double) x->size);
for (ju = 0; ju < x->size; ju++)
    u->data[ju] = - F + 2.*F * (ju / ((double) x->size - 1.));
phi_cov = malloc(cdo->n_comp * sizeof(dcomplex));
losses = malloc(t->size * sizeof(double *));
for (jt = 0; jt < t->size; jt++)
{
    pnl_vect_complex_set_dcomplex(fft_in, CZERO);
    for (ju = 0; ju < x->size; ju++)
    {
        for (jn = 0; jn < cdo->n_comp; jn++)
        {
            phi_cov[jn] = PHI_COV(jn, u->data[ju] * cdo->C[jn]->nominal);
        }
        for (iv = 0; iv < cop->size; iv++)
        {
            prod1 = CZERO;
            for (jv = 0; jv < cop->size; jv++)
            {
                prod = CONE;
                for (jn = 0; jn < cdo->n_comp; jn++)
                {
                    prod = Cmul(prod,
                                CRadd(RCmul(cp->p[jn][jt][iv + jv * cop->size]
                                             CRadd(phi_cov[jn], - 1.)),
                                         1.0));
                }
                prod1 = Cadd(prod1, RCmul(cop->weights[jv + cop->size], prod))
            }
            pnl_vect_complex_set(fft_in, ju,
                                Cadd(pnl_vect_complex_get(fft_in, ju),
                                      RCmul(cop->weights[iv], prod1)));
        }
    }
}

```

```

    }
}
pnl_fft(fft_in, fft_out);
losses[jt] = malloc(x->size * sizeof(double));
for (ju = 0; ju < x->size; ju++)
{
    losses[jt][ju] = (C / ((double) x->size - 1.)) *
        Creal(Cmul(CIexp(F * x->data[ju]),
            pnl_vect_complex_get(fft_out, ju)))
        * 2.*F / (M_2PI * ((double) x->size - 1.));
}
}
pnl_vect_complex_free(&fft_in);
pnl_vect_complex_free(&fft_out);
free(phi_cov);
free_grid(u);
return (losses);
}

```

```

int CALC(LaurentGregory)(void *Opt, void *Mod, PricingMethod *Met)
{
    PnlVect          *nominal, *intensity, *dates, *x_rates, *y_rates;
    int              n_dates, n_rates, n_tranches, t_method, is_homo;
    int              t_copula, t_recovery;
    PremiaEnumMember *e;
    double           *p_copula;
    double           *p_recovery;

    int *p_method;
    TYPEOPT *ptOpt = (TYPEOPT *) Opt;
    TYPEMOD *ptMod = (TYPEMOD *) Mod;

    premia_interf_price_cdo(ptOpt, ptMod, Met,
        &nominal, &intensity,
        &n_rates, &x_rates, &y_rates,
        &n_dates, &dates, &n_tranches,
        &p_method, &is_homo);
}

```



```

t_copula = (ptMod->t_copula.Val.V_ENUM.value);
e = lookup_premia_enum(&(ptMod->t_copula), t_copula);
p_copula = e->Par[0].Val.V_PNLVECT->array;
t_method = (is_homo ? T_METHOD_LAURENT_GREGORY_HOMO : T_METHOD_LAURENT_GREGORY);
t_recovery = (ptMod->t_recovery.Val.V_ENUM.value);
p_recovery = get_t_recovery_arg(&(ptMod->t_recovery));

price_cdo(& (ptMod->Ncomp.Val.V_PINT),
          nominal->array,
          n_dates,
          dates->array,
          n_tranches + 1, /* size of the next array */
          ptOpt->tranch.Val.V_PNLVECT->array,
          intensity->array,
          n_rates,
          x_rates->array,
          y_rates->array,
          & (t_recovery), /*t_recovery*/
          p_recovery, /* recovery params */
          & (ptMod->t_copula.Val.V_ENUM.value), /*t_copula*/
          p_copula,
          &t_method,
          p_method,
          Met->Res[0].Val.V_PNLVECT->array,
          Met->Res[1].Val.V_PNLVECT->array,
          Met->Res[2].Val.V_PNLVECT->array
        );

pnl_vect_free(&nominal);
pnl_vect_free(&intensity);
pnl_vect_free(&dates);
pnl_vect_free(&x_rates);
pnl_vect_free(&y_rates);
free(p_method);
p_method = NULL;

return OK;
}

static int CHK_OPT(LaurentGregory)(void *Opt, void *Mod)

```

```

{
    Option *ptOpt = (Option *) Opt;
    if (strcmp(ptOpt->Name, "CDO") == 0) return OK;
    return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    TYPEOPT *ptOpt = (TYPEOPT *) Opt->TypeOpt;
    int      n_tranch;
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->Par[0].Val.V_INT = 4;
        n_tranch = ptOpt->tranch.Val.V_PNLVECT->size - 1;
        Met->Res[0].Val.V_PNLVECT = pnl_vect_create_from_double(n_tranch, 0.);
        Met->Res[1].Val.V_PNLVECT = pnl_vect_create_from_double(n_tranch, 0.);
        Met->Res[2].Val.V_PNLVECT = pnl_vect_create_from_double(n_tranch, 0.);
    }

    return OK;
}

PricingMethod MET(LaurentGregory) =
{
    "Laurent_Gregory",
    { {"N subdivisions", INT, {4}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(LaurentGregory),
    { {"Price(bp)", PNLVECT, {100}, FORBID},
      {"D_leg", PNLVECT, {100}, FORBID},
      {"P_leg", PNLVECT, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(LaurentGregory),
    CHK_ok,
    MET(Init)
};

```

