

## [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"

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

static double      pp(double      x, double      y)
{
    return ((x > y) ? (x - y) : 0.);
}

double ** *Uoptimal(const CDO *cdo, const copula *cop, const grid *t, const cond
{

    double      eps      = 0.00001;
    int      Max_iter = 20;
    int      jtr;
    int      jt;
```

```

int      k      = 1;
double   u_sad1;
double   u_sad2;
int      jv, jn;
double ***U;
double   l, a, b, x, pr;
double   psi1, psi2;

U = malloc((cdo->n_tranches - 1) * sizeof(double **));

for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
{
    U[jtr] = malloc((t->size) * sizeof(double *));
}

for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
{
    for (jt = 0; jt < t->size; jt++)
    {
        U[jtr][jt] = malloc((cop->size) * sizeof(double));
    }
}

for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
{
    for (jt = 0; jt < t->size; jt++)
    {
        for (jv = 0; jv < cop->size; jv++)
        {
            U[jtr][jt][jv] = 0;
        }
    }
}

for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
{
    for (jt = 0; jt < t->size; jt++)
    {

```

```

for (jv = 0; jv < cop->size; jv++)
{
    if (cdo->tr[jtr + 1] < (1. - cdo->C[0]->mean_delta))
    {
        k = 0;

        l = (1. - cdo->C[0]->mean_delta) * (cdo->C[0]->nominal);
        pr = cp->p[0][jt][jv];

        x = (1. / l) * log(fabs((pr) * (1 * cdo->n_comp - cdo->tr[jtr + 1])))

        u_sad1 = u_sad2 = x;
        do
        {
            psi1 = 0;
            psi2 = 0;

            for (jn = 0; jn < cdo->n_comp; jn++)
            {
                l = (1. - cdo->C[jn]->mean_delta) * (cdo->C[jn]->nominal);
                a = cp->p[jn][jt][jv] * l * exp(-u_sad1 * l);
                b = 1 - cp->p[jn][jt][jv] + cp->p[jn][jt][jv] * exp(-u_sad1 * l);
                psi1 -= a / b;
                psi2 += (1 - cp->p[jn][jt][jv]) * l * a / (b * b);
            }

            u_sad2 = u_sad1 - (psi1 - (2 / u_sad1) + cdo->tr[jtr + 1]) / k;
            k++;
            if (fabs(u_sad2 - u_sad1) < eps) break;
            if (u_sad1 * u_sad2 > 0.0) u_sad1 = u_sad2;
            else if (u_sad1 * u_sad2 <= 0.0) u_sad1 = u_sad2 * 0.1;
        }
        while ((k < Max_iter));

        U[jtr][jt][jv] = u_sad2;
    }
}
}
}

```

```

    return U;
}

double      **saddlepoint(const CDO      *cdo,
                          const copula    *cop,
                          const grid      *t,
                          const cond_prob *cp,
                          double          ***U)
{
    double **sd;
    int     jv;
    int     jt;
    int     jtr;
    int     jn;

    double l, a, b, a1, b1;
    double K, c, g, psi_0, psi_2, psi_4, psi_6;

    sd = malloc((cdo->n_tranches - 1) * sizeof(double *));

    for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
    {
        sd[jtr] = malloc((t->size) * sizeof(double));

    }

    for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
    {

        for (jt = 0; jt < t->size; jt++)
        {
            sd[jtr][jt] = 0;
            for (jv = 0; jv < cop->size; jv++)
            {
                psi_0 = 0;
                psi_2 = 0;
                c = 0;
                psi_4 = 0;
                psi_6 = 0;
                if (U[jtr][jt][jv] < 0)

```

```

{
    for (jn = 0; jn < cdo->n_comp; jn++)
    {
        l = (1. - cdo->C[jn]->mean_delta) * (cdo->C[jn]->nominal);
        c += l * cp->p[jn][jt][jv];
        psi_0 += log((1 - cp->p[jn][jt][jv]) + cp->p[jn][jt][jv] *
        a = cp->p[jn][jt][jv] * l * exp(-U[jtr][jt][jv] * l);
        b = 1 - cp->p[jn][jt][jv] + cp->p[jn][jt][jv] * exp(-U[jtr][jt][jv] * l);
        K = cp->p[jn][jt][jv] * (1 - cp->p[jn][jt][jv]) * l * l;
        psi_4 += K * l * l * exp(-U[jtr][jt][jv] * l) / (b * b) -
        psi_2 += (1 - cp->p[jn][jt][jv]) * l * a / (b * b);
        psi_6 += ((1 - cp->p[jn][jt][jv]) * pow(l, 6) * cp->p[jn][jt][jv]);

    }

    a1 = exp(U[jtr][jt][jv] * cdo->tr[jtr + 1] + psi_0 - 2 * log(-
    b1 = sqrt(2 * 3.14 * (psi_2 + (2 * 1. / (U[jtr][jt][jv] * U[jtr][jt][jv])));
    g = psi_2 + 2 / (U[jtr][jt][jv] * U[jtr][jt][jv]);

    sd[jtr][jt] += (a1 * (1. / b1) * (1 + (psi_4 + 12 / (U[jtr][jt][jv] * U[jtr][jt][jv])));

}

else
{
    for (jn = 0; jn < cdo->n_comp; jn++)
    {
        l = (1. - cdo->C[jn]->mean_delta) * (cdo->C[jn]->nominal);
        psi_0 += log((1 - cp->p[jn][jt][jv]) + cp->p[jn][jt][jv] *
        a = cp->p[jn][jt][jv] * l * exp(-U[jtr][jt][jv] * l);
        b = 1 - cp->p[jn][jt][jv] + cp->p[jn][jt][jv] * exp(-U[jtr][jt][jv] * l);
        K = cp->p[jn][jt][jv] * (1 - cp->p[jn][jt][jv]) * l * l;
        psi_4 += K * l * l * exp(-U[jtr][jt][jv] * l) / (b * b) -
        psi_2 += (1 - cp->p[jn][jt][jv]) * l * a / (b * b);
        psi_6 += ((1 - cp->p[jn][jt][jv]) * pow(l, 6) * cp->p[jn][jt][jv]);

    }

    a1 = exp(U[jtr][jt][jv] * cdo->tr[jtr + 1] + psi_0 - 2 * log(U
    b1 = sqrt(2 * 3.14159265 * (psi_2 + (2 * 1. / (U[jtr][jt][jv] * U[jtr][jt][jv])));

```

```

        g = psi_2 + 2 / (U[jtr][jt][jv] * U[jtr][jt][jv]);
        sd[jtr][jt] += a1 * (1. / b1) * (1 + (psi_4 + 12 / (U[jtr][jt]
    }
}
}

return (sd);

}

double      *payment_leg_sadd(const CDO      *cdo,
                             const step_fun *rates,
                             const grid      *t,
                             double          *const *saddlepoint,
                             const copula *cop)
{
    int      jt;
    double    tjt;
    double    *pls;
    int      jpls;
    double    tau;
    double    *tab;
    double    t_previous;
    double    **numdefr = NULL;
    cond_prob *cpr      = NULL;
    int      jr;
    double    ml;
    int      jt_payment;
    double    m;
    double    ml_previous;
    double    ml2;
    double    B;
    double    tau1;

```

```

cpr = init_cond_prob(cdo, cop, t);
numdefr = lg_numdef(cdo, cop, t, cpr);

tab = malloc((cdo->n_tranches) * sizeof(double));
pls = malloc((cdo->n_tranches - 1) * sizeof(double));

for (jpls = 0; jpls < cdo->n_tranches - 1; jpls++)
{
    pls[jpls] = 0;
    tab[jpls] = 0;
}
tab[cdo->n_tranches - 1] = 0;

for (jpls = 0; jpls < cdo->n_tranches - 1; jpls++)
{
    t_previous = 0;

    if (cdo->tr[jpls + 1] < (1. - cdo->C[0]->mean_delta))
    {
        for (jt = 0; jt < t->size; jt++)
        {
            tjt = t->data[jt];
            tau = exp(- compute_sf(rates, tjt));
            tab[jpls + 1] += tau * (saddlepoint[jpls][jt]) * (tjt - t_previous);
            t_previous = tjt;
        }
    }

    else
    {
        m = cdo->C[0]->nominal * (1. - cdo->C[0]->mean_delta);
        ml_previous = 0.;
        jt_payment = 0;
        t_previous = 0;
        B = cdo->tr[jpls + 1];
    }
}

```

```

    for (jt = 0; jt < t->size; jt++)
    {
        ml = 0;
        for (jr = 0; jr < cdo->n_comp + 1; jr++)
        {
            ml += (pp(m * jr, 0.0) - pp(m * jr, B)) * numdefr[jt][jr];
        }

        ml2 = ml - ml_previous;
        ml_previous = ml;
        tjt = t->data[jt];
        tau = exp(- compute_sf(rates, tjt));
        if (tjt == cdo->dates->data[jt_payment])
        {
            tab[jpls + 1] += tau * (cdo->tr[jpls + 1] - ml) * (tjt - t_previous);
            t_previous = cdo->dates->data[jt_payment];
            jt_payment++;
        }

        tau1 = exp(- compute_sf(rates, t->data[jt] - t->delta[jt] * 0.5));
        tab[jpls + 1] += tau1 * ml2 * (tjt - t_previous);
    }
}

for (jpls = 0; jpls < cdo->n_tranches - 1; jpls++)
{
    pls[jpls] = tab[jpls + 1] - tab[jpls];
}
for (jt = 0; jt < t->size; jt++)
    free(numdefr[jt]);
free(numdefr);

free_cond_prob(cpr);
free(tab);
return (pls);
}

```



```

double          *default_leg_sadd(const CDO      *cdo,
                                const step_fun *rates,
                                const grid      *t,
                                double *const *saddlepoint,
                                const copula *cop)
{
    int          jt;
    double       t_jt;
    double       *dls;
    int          jdls;
    double       tau;
    double       *tab;
    double       t_previous;
    double       r;
    double       **numdefr = NULL ;
    cond_prob    *cpr = NULL ;

    int          jr;
    double       ml;
    double       ml_previous;
    double       m;
    double       ml2;
    double       B;
    r = compute_sf(rates, 1);

    cpr = init_cond_prob(cdo, cop, t);
    numdefr = lg_numdef(cdo, cop, t, cpr);

    dls = malloc((cdo->n_tranches - 1) * sizeof(double));
    tab = malloc((cdo->n_tranches) * sizeof(double));

    for (jdls = 0; jdls < cdo->n_tranches - 1; jdls++)
    {
        dls[jdls] = 0;
        tab[jdls] = 0;
    }

    tab[cdo->n_tranches - 1] = 0;

    for (jdls = 0; jdls < cdo->n_tranches - 1; jdls++)

```

```

{
    if (cdo->tr[jdls + 1] < (1 - cdo->C[0]->mean_delta))
    {
        tab[jdls + 1] = cdo->tr[jdls + 1] - exp(-r * (t->data[t->size - 1])) *
        t_previous = 0;

        for (jt = 0; jt < t->size; jt++)
        {
            t_jt = t->data[jt];
            tau = exp(- compute_sf(rates, t_jt));
            tab[jdls + 1] -= r * tau * (saddlepoint[jdls][jt]) * (t_jt - t_pre
            t_previous = t_jt;
        }
    }
    else
    {
        ml_previous = 0.;
        B = cdo->tr[jdls + 1];
        m = cdo->C[0]->nominal * (1. - cdo->C[0]->mean_delta);
        for (jt = 0; jt < t->size; jt++)
        {
            ml = 0;
            for (jr = 0; jr < (cdo->n_comp + 1); jr++)
            {
                ml += (pp(m * jr, 0.0) - pp(m * jr, B)) * numdefr[jt][jr];
            }
            ml2 = ml - ml_previous;
            ml_previous = ml;
            tau = exp(- compute_sf(rates, t->data[jt] - t->delta[jt] * 0.5));
            tab[jdls + 1] += tau * ml2;
        }
    }
}

for (jdls = 0; jdls < cdo->n_tranches - 1; jdls++)
{
    dls[jdls] = fabs(tab[jdls + 1] - tab[jdls]);
}

```

```

for (jt = 0; jt < t->size; jt++)
    free(numdefr[jt]);
free(numdefr);
free_cond_prob(cpr);
free(tab);
return dls;
}

int CALC(Saddlepoint)(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, *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_method = T_METHOD_SADDLEPOINT;

    t_copula = (ptMod->t_copula.Val.V_ENUM.value);
    /* Clayton and Student copula not treated */
    if (t_copula == T_COPULA_STUDENT || t_copula == T_COPULA_CLAYTON)
        return PREMIA_UNTREATED_COPULA;
    e = lookup_premia_enum(&(ptMod->t_copula), t_copula);
    p_copula = e->Par[0].Val.V_PNLVECT->array;
    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,
        p_recovery,
        &(ptMod->t_copula.Val.V_ENUM.value),
        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(Saddlepoint)(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(Saddlepoint) =
{
    "Saddlepoint",
    { {"N subdivisions", INT, {4}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(Saddlepoint),
    { {"Price(bp)", PNLVECT, {100}, FORBID},
      {"D_leg", PNLVECT, {100}, FORBID},
      {"P_leg", PNLVECT, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(Saddlepoint),
    CHK_ok,
    MET(Init)
};

```