

Help

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "copula_stdndc.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "math/cdo/cdo.h"
#include "price_cdo.h"

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

static int      compute_default(const CDO      *cdo,
                                copula      *cop,
                                int          *ind,
                                double      *tau)
{
    double      tau_jn;
    int      n_def;
    int      jn;
    int      jk;

    cop->generate(cop);

```

```

n_def = 0;
for (jn = 0; jn < cdo->n_comp; jn++)
{
    if (cop->compute_default_time(cop, cdo->C[jn]->H, &tau_jn))
    {
        jk = n_def - 1;
        while ((jk >= 0) && (tau_jn < tau[jk]))
        {
            ind[jk + 1] = ind[jk];
            tau[jk + 1] = tau[jk];
            jk--;
        }
        ind[jk + 1] = jn;
        tau[jk + 1] = tau_jn;
        n_def++;
    }
}

return (n_def);
}

double          *mc_default_one_leg(const CDO      *cdo,
                                   copula          *cop,
                                   const step_fun *rates,
                                   int              *ind,
                                   double          *tau)
{
    double      *dl;
    double      *phi_losses;
    double      losses;
    double      act;
    double      new_phi_losses;
    int         n_def;
    int         jk;
    int         jtr;

    n_def = compute_default(cdo, cop, ind, tau);
    dl = malloc((cdo->n_tranches - 1) * sizeof(double));
    phi_losses = malloc((cdo->n_tranches - 1) * sizeof(double));
    for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
    {

```

```

        dl[jtr] = 0;
        phi_losses[jtr] = 0.;
    }
    losses = 0;
    for (jk = 0; jk < n_def; jk++)
    {
        losses += cdo->C[ind[jk]]->nominal * (1. - RECOVERY(ind[jk]));
        act = exp(- compute_sf(rates, tau[jk]));
        for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
        {
            new_phi_losses = pp(losses, cdo->tr[jtr]) - pp(losses, cdo->tr[jtr + 1]);
            dl[jtr] += act * (new_phi_losses - phi_losses[jtr]);
            phi_losses[jtr] = new_phi_losses;
        }
    }
    free(phi_losses);

    return (dl);
}

double          *mc_payment_one_leg(const CDO      *cdo,
                                   copula          *cop,
                                   const step_fun *rates,
                                   int              *ind,
                                   double          *tau)
{
    double        losses;
    double        *pl;
    double        act;
    double        new_phi_losses;
    double        *phi_losses;
    double        t;
    int           jtr;
    int           n_def;
    int           jk;
    int           jt;

    n_def = compute_default(cdo, cop, ind, tau);
    pl = malloc((cdo->n_tranches - 1) * sizeof(double));
    phi_losses = malloc((cdo->n_tranches - 1) * sizeof(double));
    for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)

```

```

    {
        pl[jtr] = 0;
        phi_losses[jtr] = 0.;
    }
    losses = 0;
    jk = 0;
    t = 0;
    for (jt = 0; jt < cdo->dates->size; jt++)
    {
        while ((tau[jk] >= t) && (tau[jk] < cdo->dates->data[jt]) && (jk < n_def))
        {
            losses += cdo->C[ind[jk]]->nominal * (1. - RECOVERY(ind[jk]));
            jk++;
        }
        act = exp(- compute_sf(rates, cdo->dates->data[jt]));
        for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
        {
            new_phi_losses = pp(losses, cdo->tr[jtr]) - pp(losses, cdo->tr[jtr + 1]);
            pl[jtr] += act * (cdo->tr[jtr + 1] - cdo->tr[jtr] - new_phi_losses) *
        }
        t = cdo->dates->data[jt];
    }
    losses = 0;
    jt = 0;
    for (jk = 0; jk < n_def; jk++)
    {
        while (tau[jk] > cdo->dates->data[jt]) jt++;
        t = (jt == 0) ? 0. : cdo->dates->data[jt - 1];
        losses += cdo->C[ind[jk]]->nominal * (1. - RECOVERY(ind[jk]));
        act = exp(- compute_sf(rates, tau[jk]));
        for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
        {
            new_phi_losses = pp(losses, cdo->tr[jtr]) - pp(losses, cdo->tr[jtr + 1]);
            pl[jtr] += act * (new_phi_losses - phi_losses[jtr]) * (tau[jk] - t);
            phi_losses[jtr] = new_phi_losses;
        }
    }
    free(phi_losses);

    return (pl);
}

```

```

double          *mc_default_vc_one_leg(const CDO      *cdo,
                                       copula          *cop,
                                       const step_fun *rates,
                                       int             *ind,
                                       double          *tau)
{
    double      *dl;
    double      *phi_losses;
    double      *phi_losses_vc;
    double      losses;
    double      losses_vc;
    double      act;
    double      new_phi_losses;
    double      new_phi_losses_vc;
    double      nominal;
    double      delta;
    int         n_def;
    int         jk;
    int         jtr;
    int         jc;

    n_def = compute_default(cdo, cop, ind, tau);
    dl = malloc((cdo->n_tranches - 1) * sizeof(double));
    phi_losses = malloc((cdo->n_tranches - 1) * sizeof(double));
    phi_losses_vc = malloc((cdo->n_tranches - 1) * sizeof(double));
    for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
    {
        dl[jtr] = 0;
        phi_losses[jtr] = 0.;
        phi_losses_vc[jtr] = 0.;
    }

    nominal = 0;
    delta = 0;
    for (jc = 0; jc < cdo->n_comp; jc++)
    {
        nominal += cdo->C[jc]->nominal;
        delta += cdo->C[jc]->mean_delta;
    }
    nominal /= (double) cdo->n_comp;

```

```

delta /= (double) cdo->n_comp;

losses = 0;
losses_vc = 0;
for (jk = 0; jk < n_def; jk++)
{
    losses += cdo->C[ind[jk]]->nominal * (1. - RECOVERY(ind[jk]));
    losses_vc += nominal * (1. - delta);
    act = exp(- compute_sf(rates, tau[jk]));
    for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
    {
        new_phi_losses = pp(losses, cdo->tr[jtr]) - pp(losses, cdo->tr[jtr + 1]);
        new_phi_losses_vc = pp(losses_vc, cdo->tr[jtr]) - pp(losses_vc, cdo->tr[jtr + 1]);
        dl[jtr] += act * (new_phi_losses - phi_losses[jtr] - (new_phi_losses_vc - phi_losses_vc[jtr]));
        phi_losses[jtr] = new_phi_losses;
        phi_losses_vc[jtr] = new_phi_losses_vc;
    }
}
free(phi_losses);
free(phi_losses_vc);

return (dl);
}

double
*mc_payment_vc_one_leg(const CDO *cdo,
                        const copula *cop,
                        const step_fun *rates,
                        int ind,
                        double tau)
{
    double losses;
    double losses_vc;
    double *pl;
    double act;
    double new_phi_losses;
    double new_phi_losses_vc;
    double *phi_losses;
    double *phi_losses_vc;
    double t;
    double nominal;
    double delta;

```

```

int          jtr;
int          n_def;
int          jk;
int          jt;
int          jc;

n_def = compute_default(cdo, cop, ind, tau);
pl = malloc((cdo->n_tranches - 1) * sizeof(double));
phi_losses = malloc((cdo->n_tranches - 1) * sizeof(double));
phi_losses_vc = malloc((cdo->n_tranches - 1) * sizeof(double));
for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
{
    pl[jtr] = 0;
    phi_losses[jtr] = 0.;
    phi_losses_vc[jtr] = 0.;
}

nominal = 0;
delta = 0;
for (jc = 0; jc < cdo->n_comp; jc++)
{
    nominal += cdo->C[jc]->nominal;
    delta += cdo->C[jc]->mean_delta;
}
nominal /= (double) cdo->n_comp;
delta /= (double) cdo->n_comp;

losses = 0;
losses_vc = 0;
jk = 0;
t = 0;
for (jt = 0; jt < cdo->dates->size; jt++)
{
    while ((tau[jk] >= t) && (tau[jk] < cdo->dates->data[jt]) && (jk < n_def))
    {
        losses += cdo->C[ind[jk]]->nominal * (1. - RECOVERY(ind[jk]));
        losses_vc += nominal * (1. - delta);
        jk++;
    }
    act = exp(- compute_sf(rates, cdo->dates->data[jt]));
    for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)

```

```

        {
            new_phi_losses = pp(losses, cdo->tr[jtr]) - pp(losses, cdo->tr[jtr + 1]);
            new_phi_losses_vc = pp(losses_vc, cdo->tr[jtr]) - pp(losses_vc, cdo->tr[jtr + 1]);
            pl[jtr] += act * (new_phi_losses_vc - new_phi_losses) * (cdo->dates->date[jtr] - cdo->dates->date[jtr + 1]);
        }
        t = cdo->dates->date[jt];
    }
    losses = 0;
    losses_vc = 0;
    jt = 0;
    for (jk = 0; jk < n_def; jk++)
    {
        while (tau[jk] > cdo->dates->date[jt]) jt++;
        t = (jt == 0) ? 0. : cdo->dates->date[jt - 1];
        losses += cdo->C[ind[jk]]->nominal * (1. - RECOVERY(ind[jk]));
        losses_vc += nominal * (1. - delta);
        act = exp(- compute_sf(rates, tau[jk]));
        for (jtr = 0; jtr < cdo->n_tranches - 1; jtr++)
        {
            new_phi_losses = pp(losses, cdo->tr[jtr]) - pp(losses, cdo->tr[jtr + 1]);
            new_phi_losses_vc = pp(losses_vc, cdo->tr[jtr]) - pp(losses_vc, cdo->tr[jtr + 1]);
            pl[jtr] += act * (new_phi_losses - phi_losses[jtr] - (new_phi_losses_vc - phi_losses_vc[jtr]));
            phi_losses[jtr] = new_phi_losses;
            phi_losses_vc[jtr] = new_phi_losses_vc;
        }
    }
    free(phi_losses);
    free(phi_losses_vc);

    return (pl);
}

double mc_generic_leg(const CDO *cdo,
                      const copula *cop,
                      const step_fun *rates,
                      const int n_mc,
                      mc_one_leg *one_leg)
{
    double *leg;
    double **stock;
    double *tau;

```



```

int      *ind;
int      jnc;
int      jmc;
int      jtr;
int      ntr = cdo->n_tranches - 1;

leg = malloc(2 * (ntr) * sizeof(double));
stock = malloc(n_mc * sizeof(double *));
tau = malloc(cdo->n_comp * sizeof(double));
ind = malloc(cdo->n_comp * sizeof(int));
for (jtr = 0; jtr < 2 * ntr; jtr++)
    leg[jtr] = 0;
for (jnc = 0; jnc < cdo->n_comp; jnc++)
{
    tau[jnc] = 0;
    ind[jnc] = 0;
}
for (jmc = 0; jmc < n_mc; jmc++)
{
    stock[jmc] = one_leg(cdo, cop, rates, ind, tau);
    for (jtr = 0; jtr < ntr; jtr++)
        leg[jtr] += stock[jmc][jtr];
}
for (jtr = 0; jtr < ntr; jtr++)
    leg[jtr] /= (double) n_mc;
for (jmc = 0; jmc < n_mc; jmc++)
{
    for (jtr = 0; jtr < ntr; jtr++)
        leg[ntr + jtr] += (stock[jmc][jtr] - leg[jtr]) * (stock[jmc][jtr] - leg[jtr]);
}
for (jtr = 0; jtr < ntr; jtr++)
    leg[ntr + jtr] /= ((double) n_mc - 1.);
free(ind);
free(tau);

/* every entry of stock is actually an array */
for (jmc = 0; jmc < n_mc; jmc++) free(stock[jmc]);
free(stock);

return (leg);
}

```

```

double          *mc_default_leg(const CDO      *cdo,
                                copula         *cop,
                                const step_fun *rates,
                                const int      n_mc)
{
    return (mc_generic_leg(cdo, cop, rates, n_mc, mc_default_one_leg));
}

double          *mc_payment_leg(const CDO      *cdo,
                                copula         *cop,
                                const step_fun *rates,
                                const int      n_mc)
{
    return (mc_generic_leg(cdo, cop, rates, n_mc, mc_payment_one_leg));
}

double          *mc_default_vc_leg(const CDO      *cdo,
                                    copula         *cop,
                                    const step_fun *rates,
                                    const int      n_mc)
{
    return (mc_generic_leg(cdo, cop, rates, n_mc, mc_default_vc_one_leg));
}

double          *mc_payment_vc_leg(const CDO      *cdo,
                                    copula         *cop,
                                    const step_fun *rates,
                                    const int      n_mc)
{
    return (mc_generic_leg(cdo, cop, rates, n_mc, mc_payment_vc_one_leg));
}

int CALC(MonteCarlo)(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_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 = (Met->Par[1].Val.V_ENUM.value == 1 ? T_METHOD_MC_CV : T_METHOD_MC);
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(MonteCarlo)(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 = 10000;
        Met->Par[1].Val.V_ENUM.value = 0;
        Met->Par[1].Val.V_ENUM.members = &PremiaEnumBool;
        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(MonteCarlo) =
{
    "Monte_Carlo",
    { {"N simulations", INT, {100000}, ALLOW},

```

```
    {"Use control variate", ENUM, {1}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CALC(MonteCarlo),
{ {"Price(bp)", PNLVECT, {100}, FORBID},
  {"D_leg", PNLVECT, {100}, FORBID},
  {"P_leg", PNLVECT, {100}, FORBID},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CHK_OPT(MonteCarlo),
CHK_ok,
MET(Init)
};
```