

[Help](#)

```
#ifndef _CDO_H
#define _CDO_H

#include "
href../../../../common/math/cdo/cdo_math_h_src.pdf cdo_math.h"
#include "
href../../../../common/math/cdo/structs_h_src.pdf structs.h"
#include "
href../../../../common/math/cdo/company_h_src.pdf company.h"
#include "
href../../../../common/math/cdo/copulas_h_src.pdf copulas.h"
#define PHI_COV(j, u) ( cdo->C[j]->phi_recov(u, cdo->C[j]->p_recovery) )
#define RECOVERY(j) ( cdo->C[j]->generate_delta(cdo->C[j]->p_recovery) )

/*
 * List of pricing methods for CDO
 */
enum
{
    T_METHOD_LAURENT_GREGORY_HOMO,
    T_METHOD_LAURENT_GREGORY,
    T_METHOD_HULL_WHITE_HOMO,
    T_METHOD_HULL_WHITE,
    T_METHOD_STEIN_HOMO,
    T_METHOD_STEIN,
    T_METHOD_SADDLEPOINT,
    T_METHOD_MC,
    T_METHOD_MC_CV
};

/*
 * List of Copulas
 */
enum
{
    T_COPULA_GAUSS = 1,
    T_COPULA_CLAYTON = 2,
    T_COPULA_NIG = 3,
```

```

    T_COPULA_STUDENT = 4,
    T_COPULA_DOUBLE_T = 5
};

/*
 * List of possible recovery types
 */
enum
{
    T_RECOVERY_CONSTANT = 1,
    T_RECOVERY_UNIFORM = 2,
    T_RECOVERY_GAUSS = 3 /* not interfaced sofar */
};

typedef struct
{
    int      n_comp;
    company **C;
    grid     *dates;
    int      n_tranches;
    double   *tr;
} CDO;

typedef struct
{
    int      nb;                /**nombre d'entreprise;**/
    double   recov;             /**le Recovery de l'entreprise;**/
    double   nominal;          /**le nominal de l'entreprise;**/
    int      maturite;          /**la maturité du contrat;**/
    double   att;               /**point d'attachement**/
    double   det;               /** point de détachement**/
    double   rate;              /**taux d'interet**/
} prod;

CDO      *init_CDO(const int      n_comp,
                  company      **C,
                  const int      n_dates,
                  const double   *t,
                  const int      n_tranches,
                  const double   *tr);

```

```

CDO *homogenize_CDO(const CDO    *cdo);

void free_cdo(CDO                **cdo);

typedef struct
{
    double ** *p;
    int      n_comp;
    int      n_t;
} cond_prob;

cond_prob  *init_cond_prob(const CDO    *cdo,
                           const copula *cop,
                           const grid   *t);

void free_cond_prob(cond_prob          *cp);

grid        **mean_losses(const CDO    *cdo,
                           const grid   *t,
                           const grid   *x,
                           double *const *losses);

grid        **mean_losses_from_numdef(const CDO    *cdo,
                                       const grid   *t,
                                       double *const *numdef);

double       *payment_leg(const CDO    *cdo,
                           const step_fun *rates,
                           const grid   *x,
                           grid *const  *mean_losses);

double       *default_leg(const CDO    *cdo,
                           const step_fun *rates,
                           const grid   *x,
                           grid *const  *mean_losses);

double       **lg_numdef(const CDO    *cdo,
                          const copula *cop,
                          const grid   *t,
                          const cond_prob *cp);

```

```

double      **lg_numdef1(const CDO      *cdo,
                        const copula    *cop,
                        const grid      *t,
                        const cond_prob *cp);

double      **lg_losses(const CDO      *cdo,
                        const copula    *cop,
                        const grid      *t,
                        const grid      *x,
                        const cond_prob *cp);

double      **lg_losses1(const CDO      *cdo,
                        const copula    *cop,
                        const grid      *t,
                        const grid      *x,
                        const cond_prob *cp);

double      **hw_numdef(const CDO      *cdo,
                        const copula    *cop,
                        const grid      *t,
                        const cond_prob *cp);

double      **hw_numdef1(const CDO      *cdo,
                        const copula    *cop,
                        const grid      *t,
                        const cond_prob *cp);

double      **hw_losses_h(const CDO      *cdo,
                        const copula    *cop,
                        const grid      *t,
                        const grid      *x,
                        const cond_prob *p);

double      **hw_losses_h1(const CDO      *cdo,
                        const copula    *cop,
                        const grid      *t,
                        const grid      *x,
                        const cond_prob *cp);

```

```

double      **hw_losses_nh(const CDO      *cdo,
                           const copula   *cop,
                           const grid      *t,
                           const grid      *x,
                           const cond_prob *cp);
double      **hw_losses_nh1(const CDO      *cdo,
                            const copula   *cop,
                            const grid      *t,
                            const grid      *x,
                            const cond_prob *cp);

```

```

double      **hw_losses_nh2(const CDO      *cdo,
                            const copula   *cop,
                            const grid      *t,
                            const grid      *x,
                            const cond_prob *cp);

```

```

double perte(const prod  *produit,
             const double f_t);

```

```

double cond_prob1(const double lambda,
                  const double t,
                  const double v,
                  const double rho);

```

```

/**valeur du contrat CDO pour un spread fixé et pour une proba de défaillance fi
x représente le facteur ,conditionnement à ce facteur les défauts sont indépe

```

```

double eval_contrat(const prod  *produit,
                    const double s,
                    const double rho,
                    const double lambda ,
                    const double v);

```

```

/**prix du contrat en prenant l'esperance de de eval_contrat sachant que v est u

```

```

double prix_contrat_CDO(const prod  *produit,

```

```

        const double s,
        const double rho,
        const double lambda);

double prix_contrat_CDS(const prod *produit,
        const double lambda,
        const double s);

/**Pour determiner C(a,b,v)conditionnellment à v utile pour les bases correlati

double eval_loss(const prod *produit,
        const double rho,
        const double lambda ,
        const double v);

/**Pour evaluer C(a,b) pour les bases correlations ***/

double loss(const prod *produit,
        const double rho,
        const double lambda);

/**srpead conditionnel à v**/

/**spread**/

double spread_CDO(const prod *produit,
        const double rho,
        const double lambda);

double spread_CDS(const prod *produit,
        const double lambda);

double payment_leg_CDO(const prod *produit,
        const double rho,
        const double lambda);

double payleg_cond_CDO(const prod *produit,
        const double rho,
        const double lambda ,

```

```

        const double v);

double intens1(const prod    *produit,
               const double s);

/**Autre méthode Newton +dichotomie **/

double intens2(const prod    *produit,
               const double s);

/**On suppose qu'on a les données sur les 5 tranches + l'index on cherche à dete
    chaque tranche par inversion de la formule su spread **/

double *tranche_correl(const prod    *produit,
                       const double *s1,
                       const double s2,
                       const int     choix);

/** Base correlation voir papier Hull and White perfect copula pour déterminer l
    corrélations implicites pour qu'il y'ait pas opportunité d'arbitrage **/

double *Base_correl(const prod    *produit,
                    const double *s1,
                    const double s2,
                    const int     choix);

double *lambdaimpl(const prod    *produit,
                   const double *s1,
                   const double s2,
                   const int     n,
                   const int     choix);

/** s1 tableaux des 5 spreads des tranches de CDO
    s2 valeur du spread de l'index CDS
    n nombre d'intensité implicites que l'on désire **/

```

```

/**fonction retournant la matrice du gradient de la fonction qu'on cherche à min

double **gradient(const prod    *produit,
                  const double *s1,
                  const double  s2,
                  const int     n,
                  const int     choix);

/** fonction renvoyant le spread d'une tranche de CDO par la méthode base-corre

double pay_leg_base(const prod    *produit,
                   const double *s1,
                   const double  s2,
                   const int     choix);
double dl_leg_base(const prod    *produit,
                   const double *s1,
                   const double  s2,
                   const int     choix);

double *probaimpl(const prod    *produit,
                  const double *s1,
                  const double  s2,
                  const int     n,
                  const int     choix);
double pay_leg_impl(const prod    *produit,
                   const double *p,
                   const int     n);

double dl_leg_impl(const prod    *produit,
                   const double *p,
                   const int     n);

typedef double      *(mc_one_leg)(const CDO      *cdo,
                                  const copula     *cop,
                                  const step_fun   *rates,
                                  const int         *ind,

```



```

double *tau);

double *mc_default_leg(const CDO *cdo,
                      copula *cop,
                      const step_fun *rates,
                      const int n_mc);

double *mc_payment_leg(const CDO *cdo,
                      copula *cop,
                      const step_fun *rates,
                      const int n_mc);

double *mc_default_vc_leg(const CDO *cdo,
                      copula *cop,
                      const step_fun *rates,
                      const int n_mc);

double *mc_payment_vc_leg(const CDO *cdo,
                      copula *cop,
                      const step_fun *rates,
                      const int n_mc);

/*****
Saddlepoint
*****/

double **saddlepoint(const CDO *cdo,
                    const copula *cop,
                    const grid *t,
                    const cond_prob *cp,
                    double ***U);

double *payment_leg_sadd(const CDO *cdo,
                    const step_fun *rates,
                    const grid *t,
                    double *const *saddlepoint,
                    const copula *cop);

double *default_leg_sadd(const CDO *cdo,
                    const step_fun *rates,

```

```
const grid      *t,  
double *const   *saddlepoint,  
const copula    *cop);  
  
double ** *Uoptimal(const CDO *cdo, const copula *cop, const grid *t, const cond  
  
#endif
```