

[Help](#)

```
#include <stdlib.h>
#include <
href../../common/math/cdo/cdo_math_h_src.pdfmath.h>
#include "pnl/pnl_complex.h"
#include "pnl/pnl_cdf.h"
#include "
href../../mod/static_merton/static_merton_stdr/static_merton_stdr_h_src.pdfst

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2014+2) //The "#els
static int CHK_OPT(AP_MASDEMONTORTIZ)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_MASDEMONTORTIZ)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double *y;
static int y_punt;
static double pw2n, pw2n2;
static double ext_inf, ext_sup;
static int m;
static int num_obligors;
static double *ueval, *ReQ, *E, * *pd_ciclo_mat, *PD, *Eaux, *T;
static double r_cicle;

/* Series coefficients */
static double coef(int k)
{
    int j;
    double sum, aux1, aux2;

    aux1 = ReQ[0] * cos(k * ueval[0]);
    aux2 = ReQ[m] * cos(k * ueval[m]);

    sum = aux1 + aux2;
```

```

    for (j = 1; j < m; j++) sum += 2 * ReQ[j] * cos(k * ueval[j]);

    return (sum / (m * pow(r_cicle, k)));
}

static double g(double y)
{
    return ((1. / sqrt(2 * M_PI)) * exp(-0.5 * pow(y, 2)));
}

static dcomplex prod(dcomplex w, int i)
{
    dcomplex p;
    int k;

    p = CONE;
    for (k = 0; k < num_obligors; k++)
    {
        p = Cmul(p, CRadd(CRmul(Cexp(Cmul(w, Complex(0., -E[k]))), pd_ciclo_mat[k]
                                1. - pd_ciclo_mat[k][i])));
    }

    return (p);
}

static dcomplex chf(dcomplex w)
{
    dcomplex sum;
    int i;

    sum = CZERO;
    for (i = 0; i < y_punt; i++)
    {
        sum = Cadd(sum, CRmul(prod(w, i), g(y[i]) * (y[1] - y[0])));
    }

    return (sum);
}

static void Q(double x, double res2[])
{

```

```

dcomplex z, z1, z2;

z = Complex_polar(r_cicle, x);
z1 = Csub(chf(Cmul(RCmul(pw2n, Clog(z)), CI)), Cpow_real(z, pw2n));
z2 = C_op_damb(pw2n2, CONE, z); /* z2 = pw2n2 * (1. - z) */

res2[0] = Creal(Cdiv(z1, z2));
}

static int ap_masdemontortiz(int NO, double pd, double alpha, double rho, int n,
{
    int k_max;
    int k, j, i;
    double aux, c;
    double benchmark, sum_coef;
    double result[2];
    double c_ini, c_mov, c_fin, dif, dif2;

    pw2n = pow(2, n);
    pw2n2 = sqrt(pw2n);

    k_max = pw2n - 1;

    m = pw2n;
    num_obligors = NO;

    ext_inf = 0;
    ext_sup = 1.;

    //Number of integration points
    y_punt = 100;
    //Integration parameter in the Cauchy formula
    r_cicle = 0.9995;

    /* PD and exposure, assume LGD=1 */
    y = malloc(y_punt * sizeof(double));
    E = malloc((NO) * sizeof(double));
    PD = malloc((NO) * sizeof(double));
    Eaux = malloc((NO) * sizeof(double));
    T = malloc((NO) * sizeof(double));

```

```

aux = 0;
for (i = 0; i < NO; i++)
{
    Eaux[i] = 1. / (i + 1);
    aux = aux + Eaux[i];
}
c = 1. / aux;
for (i = 0; i < NO; i++)
{
    E[i] = c * 1. / (i + 1);
    PD[i] = pd;
    T[i] = pnl_inv_cdfnor(PD[i]);
}

/* Conditional PD's */

/* Allocate memory first */

pd_ciclo_mat = (double **)calloc(NO, sizeof(double *));
for (i = 0; i < NO; i++)
{
    pd_ciclo_mat[i] = (double *)calloc(y_punt, sizeof(double));
}

for (i = 0; i < y_punt; i++) y[i] = -5. + 0.1 * i;

for (j = 0; j < NO; j++)
    for (i = 0; i < y_punt; i++)
        pd_ciclo_mat[j][i] = pnl_cdfnor((T[j] - sqrt(rho) * y[i]) / sqrt(1. - rho))

ReQ = malloc((m + 1) * sizeof(double));
ueval = malloc((m + 1) * sizeof(double));

for (j = 0; j <= m; j++)
{
    ueval[j] = (M_PI / m) * j;
    Q(ueval[j], result);
    ReQ[j] = result[0];
}

```

```

/* VaR computation */

c_ini = pw2n / 2;
c_fin = c_ini;
c_mov = c_ini / 2;
benchmark = alpha;
c = coef(c_ini);
dif = fabs(c * pw2n2 - benchmark);
for (j = 1; j < n; j++)
{
    if (c > alpha / pw2n2) c_ini = c_ini - c_mov;
    else c_ini = c_ini + c_mov;
    c_mov = c_mov / 2;
    c = coef(c_ini);
    dif2 = fabs(c * pw2n2 - benchmark);
    if (dif2 < dif)
    {
        c_fin = c_ini;
        dif = dif2;
    }
}

*var = (2 * c_fin + 1) / (2 * pw2n);

/* ES computation */

sum_coef = 0;
for (k = c_fin + 1; k <= k_max; k++) sum_coef += coef(k);

*expected_shortfall = (1. / (1 - alpha)) * (1 - alpha * (*var) - (1. / (2 * pw

free(ReQ);
free(ueval);
free(E);
free(Eaux);
free(PD);
free(T);
free(y);
for (i = 0; i < NO; i++)
    free(pd_ciclo_mat[i]);

```

```

    free(pd_ciclo_mat);

    return OK;
}

int CALC(AP_MASDEMONTORTIZ)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return ap_masdemontortiz(ptOpt->NumberOfCreditors.Val.V_PINT,
                             ptOpt->ConstantProbabilityofDefault.Val.V_RGDOUBLE,
                             ptMod->rho.Val.V_RGDOUBLE,
                             Met->Par[0].Val.V_PINT,
                             &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUB
    }

static int CHK_OPT(AP_MASDEMONTORTIZ)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "CreditVaRisk") == 0))
        return OK;
    return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    static int first = 1;

    if (first)
    {
        Met->HelpFilenameHint = "AP_MASDEMONTORTIZ";
        Met->Par[0].Val.V_PINT = 10;
        first = 0;
    }
    return OK;
}

PricingMethod MET(AP_MASDEMONTORTIZ) =
{
    "AP_MASDEMONTORTIZ",

```

```

{ {"Approximation Scale", INT, {100}, ALLOW},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CALC(AP_MASDEMONTORTIZ),
{ {"Value At Risk", DOUBLE, {100}, FORBID},
  {"Conditional Tail Expectation", DOUBLE, {100}, FORBID},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CHK_OPT(AP_MASDEMONTORTIZ),
CHK_split,
MET(Init)
};

```