

## [Help](#)

```
#include "
href../../../../mod/mer1d/mer1d_std/mer1d_std_h_src.pdfmer1d_std.h"
#include "
href../../../../common/error_msg_h_src.pdferror_msg.h"
#include <
href../../../../common/math/cdo/cdo_math_h_src.pdfmath.h>
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2012+2) //The "#els
static int CHK_OPT(AP_STATICHEDGING_CARRWU)(void *Opt, void *Mod)
{
    return NONACTIVE;
}

int CALC(AP_STATICHEDGING_CARRWU)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else
/* Gamma in call_merton price */
static int GammaCall_Merton(double x, double K, double T, double r, double divid
{
    double lambdaT, mv2, exmv2, EU, mu, M, sigma02, sigmasqrt, gamma, test, puissa

    lambdaT = lambda * T;
    mv2 = m + v / 2.;
    exmv2 = exp(mv2);
    EU = exmv2 - 1;
    mu = r - divid - lambda * EU;
    muT = mu * T;
    M = exp(T * (-divid - lambda * exmv2));
    sigma02 = sigma * sigma;
    sigmasqrt = sigma * sqrt(T);
    d1 = (log(x / K) + sigma02 * T / 2 + muT) / sigmasqrt;
    puissancen1 = 1.;
    factorieln = 1.;
    test = exp(-lambdaT);
    puissancen1 = 1.;
```

```

factorieln = 1.;
n = 0;
gamma = 0.;
while (test < 0.99999)
{
    n++;

    factorieln *= n; /* n! */
    puissancen1 *= lambdaT; /* (lambda*T)^n */
    sigma2 = sigma02 + v * (double)n / T;
    sigmasqrt = sqrt(sigma2 * T);
    d1 = (log(x / K) + sigma2 * T / 2 + n * (mv2) + muT) / sigmasqrt;

    test += exp(-lambdaT) * puissancen1 / factorieln;

    gamma += (puissancen1 / factorieln) * exp(n * mv2) * M * cdf_nor(d1) / (K
}
*ptgamma = gamma;

return 0;
}

static int Merton_Weights(double x, NumFunc_1 *p, double T2, double r, double d
{
    int i;
    int N_points = 5;
    double K;
    double nu, correctedsigma, tildeK, ptgamma;
    PnlVect *Pt_gh = pnl_vect_create((int) N_points); /* gauss hermite points */
    PnlVect *Wg_gh = pnl_vect_create((int) N_points); /* gauss hermite weights */

    if (T1 > T2)
        return HEGDING_MATURITY_GREATER_THAN_MATURITY;

    K = p->Par[0].Val.V_PDOUBLE;

    pnl_vect_set(Pt_gh, 0, 0.0);
    pnl_vect_set(Pt_gh, 1, 0.958572);
    pnl_vect_set(Pt_gh, 2, -0.958572);

```

```

pnl_vect_set(Pt_gh, 3, 2.02018);
pnl_vect_set(Pt_gh, 4, -2.02018);
pnl_vect_set(Wg_gh, 0, 0.945309);
pnl_vect_set(Wg_gh, 1, 0.393619);
pnl_vect_set(Wg_gh, 2, 0.393619);
pnl_vect_set(Wg_gh, 3, 0.0199532);
pnl_vect_set(Wg_gh, 4, 0.0199532);

/* Variables for the computation*/
pnl_vect_resize(Strike, (*Pt_gh).size);
pnl_vect_resize(Weights_Strike, (*Pt_gh).size);
nu = sigma * sigma + lambda * (m * m + v);
correctedsigma = sqrt(2 * nu * (T2 - T1));
tildeK = K * exp((divid - r - nu / 2) * (T2 - T1));

//Computation of the strikes and the strikes weights
for (i = 0; i < (*Pt_gh).size; i++)
{
    pnl_vect_set(Strike, i, tildeK * exp(pnl_vect_get(Pt_gh, i)*correctedsigma));

    GammaCall_Merton(x, pnl_vect_get(Strike, i), T2 - T1, r, divid, sigma, lambda);
    pnl_vect_set(Weights_Strike, i, ptgamma * pnl_vect_get(Strike, i)*correctedsigma);
}
pnl_vect_free(&Pt_gh);
pnl_vect_free(&Wg_gh);

return OK;
}

int CALC(AP_STATICHEDGING_CARRWU)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

    return Merton_Weights(ptMod->S0.Val.V_PDOUBLE, ptOpt->PayOff.Val.V_NUMFUNC_1,

```

```

}

static int CHK_OPT(AP_STATICHEDGING_CARRWU)(void *Opt, void *Mod)
{
    /*
    Option* ptOpt=(Option*)Opt;
    TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);
    */
    return strcmp(((Option *)Opt)->Name, "CallEuro");
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->Par[0].Val.V_DATE = 0.5;
        Met->Res[0].Val.V_PNLVECT = NULL;
        Met->Res[1].Val.V_PNLVECT = NULL;
    }

    /* some initialisation */
    if (Met->Res[0].Val.V_PNLVECT == NULL)
        Met->Res[0].Val.V_PNLVECT = pnl_vect_create(5);
    else
        pnl_vect_resize(Met->Res[0].Val.V_PNLVECT, 5);

    if (Met->Res[1].Val.V_PNLVECT == NULL)
        Met->Res[1].Val.V_PNLVECT = pnl_vect_create(5);
    else
        pnl_vect_resize(Met->Res[1].Val.V_PNLVECT, 5);

    return OK;
}

PricingMethod MET(AP_STATICHEDGING_CARRWU) =
{
    "AP_STATICHEDGING_CARRWU",
    {"Hedging Maturity", DATE, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID},
    CALC(AP_STATICHEDGING_CARRWU),

```

```
{{" Strikes", PNLVECT, {100}, FORBID}, {" Strikes Weights", PNLVECT, {1}, FORBID},
CHK_OPT(AP_STATICHEDGING_CARRWU),
CHK_ok,
MET(Init)
} ;
```