

[Help](#)

```
extern "C" {
#include "cirpp2d_std.h"
    extern char premia_data_dir[MAX_PATH_LEN];
    extern char *path_sep;
}

#include "math/credit_cds/cdscirpp.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
#else

static int cf_gaussmapping_cds(
    int flag_data,
    double date,
    double x0_r,
    double mrRate,
    double thetaRate,
    double sigmaRate,
    double x0,
    double mrIntensity,
    double thetaIntensity,
    double sigmaIntensity,
    double correlation,
    double maturity,
    int period,
    double recovery,
    double *spread)
{
    maturity -= date;

    std::string path(premia_data_dir);
    path += path_sep;

    std::ifstream zcb((path + "zcb.txt").c_str());

    if (!zcb)
        return UNABLE_TO_OPEN_FILE;
```

```
double T, P;

std::vector<double>    RatesMat, Rates;
std::vector<double>    intMat, intRates;

while (zcb >> T >> P)
{
    RatesMat.push_back(T);
    Rates.push_back(P);
}

if (flag_data == 0)
{
    std::ifstream intensity_file((path + "intensity.txt").c_str());

    if (!intensity_file)
        return UNABLE_TO_OPEN_FILE;

    while (intensity_file >> T >> P)
    {
        intMat.push_back(T);
        intRates.push_back(P);
    }
}
else
{
    std::ifstream cds_file((path + "cds.txt").c_str());

    if (!cds_file)
        return UNABLE_TO_OPEN_FILE;

    std::vector<double> spreadMat, spreads;

    while (cds_file >> T >> P)
    {
        spreadMat.push_back(T);
        spreads.push_back(P);
    }

    // TODO: put it to parameters.
    // What's to do with recovery and period?
```

```

    double r = 0.03;

    DefaultIntensityCalibration(recovery, period, spreadMat, spreads, r, intMa
    }

double dummy;

/*Price*/
*spread = cds_spread_GaussMap(
    maturity, // maturity of the CDS
    period, // payment period, in months
    recovery, // expected recovery rate
    mrRate, // mean reversion coefficient in the interest rate model
    mrIntensity, // mean reversion coefficient in the intensity model
    sigmaRate, // volatility coefficient in the interest rate model
    sigmaIntensity, // volatility coefficient in the intensity model
    thetaRate, // long-run mean in the interest rate model
    thetaIntensity, // long-run mean in the intensity model
    x0_r, // Starting value of the short rate process
    x0, // Starting value of the intensity process
    correlation, // correlation between rate and intensity
    RatesMat, // Maturities of zero-coupons for calibration
    Rates, // rates of risk-free zero-coupons for calibration
    intMat, // Maturities of CDS used for calibration
    intRates, // intensity of the name underlying the CDS; (spreads of
    dummy, // DefaultLeg price (return parameter)
    dummy // PaymentLeg price (return parameter)
);

    return OK;
}
#endif //PremiaCurrentVersion

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

```

```

        return AVAILABLE_IN_FULL_PREMIA;
    }
#else
    int CALC(CF_GAUSSMAPPING_CDS)(void *Opt, void *Mod, PricingMethod *Met)
    {
        TYPEOPT *ptOpt = (TYPEOPT *)Opt;
        TYPEMOD *ptMod = (TYPEMOD *)Mod;

        return cf_gaussmapping_cds(
            ptMod->flat_flag.Val.V_INT,
            ptMod->T.Val.V_DATE,
            ptMod->InitialYieldsR.Val.V_PDOUBLE,
            ptMod->aR.Val.V_DOUBLE,
            ptMod->bR.Val.V_DOUBLE,
            ptMod->SigmaR.Val.V_PDOUBLE,
            ptMod->InitialYieldsI.Val.V_PDOUBLE,
            ptMod->aI.Val.V_DOUBLE,
            ptMod->bI.Val.V_DOUBLE,
            ptMod->SigmaI.Val.V_PDOUBLE,
            ptMod->Rho.Val.V_PDOUBLE,
            ptOpt->Maturity.Val.V_DATE,
            ptOpt->NbPayement.Val.V_PINT,
            ptOpt->Recovery.Val.V_PDOUBLE,
            &(Met->Res[0].Val.V_DOUBLE));
    }

static int CHK_OPT(CF_GAUSSMAPPING_CDS)(void *Opt, void *Mod)
{
    return strcmp(((Option *)Opt)->Name, "CreditDefaultSwap");
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
    }
}

```

```
    return OK;
}

PricingMethod MET(CF_GAUSSMAPPING_CDS) =
{
    "CF_GaussianMapping_CDS",
    {{ " ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(CF_GAUSSMAPPING_CDS),
    {{ "CDS Spread", DOUBLE, {100}, FORBID}, { " ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(CF_GAUSSMAPPING_CDS),
    CHK_ok,
    MET(Init)
} ;
}
```