

## [Help](#)

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

/// \ file cdscirppmc.cpp
/// \ brief CDS_CIRpp_MC class
/// \ author M. Ciuca (MathFi, ENPC)
/// \ note (C) Copyright Premia 8 - 2006, under Premia 8 Software license
//
// Use, modification and distribution are subject to the
// Premia 8 Software license

#include <stdexcept>

#include "
href../../../../common/math/credit_cds/cdsmkt_h_src.pdfcdsmkt.h"
#include "
href../../../../common/math/credit_cds/cdscirpp_h_src.pdfcdscirpp.h"
#include "
href../../../../common/math/credit_cds/cdscirppmc_h_src.pdfcdscirppmc.h"
#include "
href../../../../common/math/credit_cds/gm_h_src.pdfgm.h"
#include "
href../../../../common/math/credit_cds/intensitycalib_h_src.pdfintensitycalib.h"

int write_plicData(double r, double Z,
                  int n, double *timesT,
                  int noCDS,
                  double arrayCDS[][2])
{
    cout << "\ n> The maximum number of calibrating CDS market data cannot "
    << "exceed: "
    << PIECEWISE_NUMBER << ".\ nTo effectively change this number, you have t
    << "\ n [1] change the value of the constant PIECEWISE_NUMBER "
    << "in the header "
    << "file \ "cds_plini.h\ "\ n [2] recompile the cpp file \ "cds_plini.cpp
    << "\ n\ n";

    cout << "> Short-rate characteristics:\ n";
}
```

```

cout << "constant r = " << r << "\ n\ n";

cout << "> Calibrating CDS market data characteristics:\ n";
cout << "Z = " << Z << endl;
cout << "We have " << noCDS << " CDS quotes (i.e. R_f's), which are as "
    << "follows:" << endl;
int i;
for (i = 0; i < noCDS; i++)
    cout << "Maturity T = " << arrayCDS[i][0] << ", R_f = " << arrayCDS[i][1]
        << endl;

cout << "R_f is paid at times: ";
for (i = 1; i <= n; i++)
    cout << " " << timesT[i];
cout << " (" << n << " payment dates)" << endl;

return 0;
}

int Write2DVector(double v[][2], int dim)
{

    int i;
    for (i = 0; i < dim; i++)
        cout << v[i][0] << " " << v[i][1] << endl;

    return 0;
}

// Very simple calibration of default intensity.
// Interest rates are flat, and
void DefaultIntensityCalibration( // Computes the implied deterministic default
    double recovery, // expected recovery rate
    int period, // payment period, in months
    std::vector<double> &spreadsMat, // Maturities of CDS used for calibration
    std::vector<double> &spreads, // spreads of CDS used for calibration
    double r, // instantaneous short rate (flat interest rates)
    std::vector<double> &intensityMat, // time grid points from the calibrated i
    std::vector<double> &intensityRates // intensity of the name underlying the C

```

```

)
{
    if (spreadsMat.size() != spreads.size())
        throw logic_error("*** Error: DefaultIntensityCalibration: spreadMat and spr

    double timesT[20];

    int noCDS = spreadsMat.size();
    double arrayCDS[10][2];
    double gamma[PIECEWISE_NUMBER + 1][2];
    int i;
    for (i = 0; i < noCDS; i++)
    {
        arrayCDS[i][0] = spreadsMat[i];
        arrayCDS[i][1] = spreads[i];
    }

    double t, yearFrac;
    t = yearFrac = (12. / period);
    int periodN = static_cast<int>(spreadsMat[spreadsMat.size() - 1] / yearFrac);
    timesT[0] = 0.0;
    for (i = 0; i < periodN; i++)
    {
        timesT[i + 1] = t;
        t += yearFrac;
    }

    cds_plini_cali(r, 1 - recovery, periodN, timesT, noCDS, arrayCDS, gamma);

    for (i = 0; i < noCDS + 1; i++)
    {
        intensityMat.push_back(gamma[i][0]);
        intensityRates.push_back(gamma[i][1]);
    }
}

double cds_spread_GaussMap( // Computes the value of the spread which correspond
    double maturity, // maturity of the CDS
    int period, // payment period, in months
    double recovery, // expected recovery rate
    double k_r, // mean reversion coefficient in the interest rate model

```

```

double k, // mean reversion coefficient in the intensity model
double sigma_r, // volatility coefficient in the interest rate model
double sigma, // volatility coefficient in the intensity model
double theta_r, // long-run mean in the interest rate model
double theta, // long-run mean in the intensity model
double x0_r, // Starting value of the short rate process
double x0, // Starting value of the intensity process
double correlation, // correlation between rate and intensity
std::vector<double> &RatesMat, // Maturities of zero-coupons for calibration
std::vector<double> &Rates, // rates of risk-free zero-coupons for calibration
std::vector<double> &intensityMat, // Maturities of CDS used for calibration
std::vector<double> &intensityRates, // intensity of the name underlying the
double &DefaultLeg, // DefaultLeg price (return parameter)
double &PaymentLeg // PaymentLeg price (return parameter)
)

{
    vector<double> timesT;

    timesT.push_back(0.0);
    double t, yearFrac;
    t = yearFrac = (1. / period);
    int periodN = static_cast<int>(maturity / yearFrac) ;
    for (int i = 0; i <= periodN; i++)
    {
        timesT.push_back(t);
        t += yearFrac;
    }

    CDS_GaussianMapping_Old cds_gm(k, theta, sigma, x0, intensityMat, intensityRates,
                                   k_r, theta_r, sigma_r, x0_r, RatesMat, Rates,
                                   correlation, timesT, 1 - recovery);

    return cds_gm.Quote(maturity, periodN, DefaultLeg, PaymentLeg);
}

double cds_spread_CIRPPMC_CV( // Computes the value of the spread which corresponds to
    double maturity, // maturity of the CDS (in years)

```

```

int period, // payment period, in months
double recovery, // expected recovery rate
double precision, // time step for CIR processes path simulation scheme
int Nsim, // number of Monte Carlo simulations
double mrRate, // mean reversion coefficient in the interest rate model
double mrIntensity, // mean reversion coefficient in the intensity model
double sigmaRate, // volatility coefficient in the interest rate model
double sigmaIntensity, // volatility coefficient in the intensity model
double thetaRate, // long-run mean in the interest rate model
double thetaIntensity, // long-run mean in the intensity model
double x0_r, // Starting value of the short rate process
double x0, // Starting value of the intensity process
double correlation, // correlation between rate and intensity
std::vector<double> &RatesMat, // Maturities of zero-coupons for calibration
std::vector<double> &Rates, // rates of risk-free zero-coupons for calibration
std::vector<double> &intensityMat, // Maturities of CDS used for calibration
std::vector<double> &intensityRates, // intensity of the name underlying the
double &DefaultLeg, // DefaultLeg price (return parameter)
double &PaymentLeg, // PaymentLeg price (return parameter)
double &std_dev_DefaultLeg, // DefaultLeg standard deviation (return parameter)
double &std_dev_PaymentLeg, // PaymentLeg standard deviation (return parameter)
double barrier, // Barrier for the intensity process
int generator
)
{

double _barrier = (barrier == 1) ? 4 * sigmaIntensity : barrier;

CDS_CIRpp_MC cds1(generator, mrIntensity, thetaIntensity, sigmaIntensity, x0,
intensityMat, intensityRates,
mrRate, thetaRate, sigmaRate, x0_r, RatesMat, Rates,
correlation, maturity, period, recovery, 1000, precision, _b

double sumI1, sumI2, sumS, b, c;
cds1.MonteCarlo(sumI1, sumI2, sumS, 1000);
//cout << "I1, I2, S: " << sumI1 << " " << sumI2 << " " << sumS << endl;

CDS_CIRpp_MC cds2(generator, mrIntensity, thetaIntensity, sigmaIntensity, x0,
intensityMat, intensityRates,
mrRate, thetaRate, sigmaRate, x0_r, RatesMat, Rates,

```

```

        correlation, maturity, period, recovery, 100, precision, _ba

cds2.Compute_b_and_c(b, c, sumI1, sumI2, sumS, 100);

CDS_CIRpp_MC cds3(generator, mrIntensity, thetaIntensity, sigmaIntensity, x0,
                    intensityMat, intensityRates,
                    mrRate, thetaRate, sigmaRate, x0_r, RatesMat, Rates,
                    correlation, maturity, period, recovery, Nsim, precision, _b

cds3.Set_b_and_c(b, c);
return cds3.CdsRate_ControlVariate(DefaultLeg, PaymentLeg, std_dev_DefaultLeg,
}

double cds_spread_CIRPPMC_MKT( // Computes the value of the spread which correspo
double maturity, // maturity of the CDS (in years)
int period, // payment period, in months
double recovery, // expected recovery rate
std::vector<double> &RatesMat, // Maturities of zero-coupons for calibration
std::vector<double> &Rates, // rates of risk-free zero-coupons for calibration
std::vector<double> &intensityMat, // Maturities of CDS used for calibration
std::vector<double> &intensityRates, // intensity of the name underlying the
double &DefaultLeg, // DefaultLeg price (return parameter)
double &PaymentLeg // PaymentLeg price (return parameter)
)
{
    CDS_NoCorr_MarketData cdsmkt(intensityMat, intensityRates,
                                RatesMat, Rates, maturity, period, recovery);

    return cdsmkt.CdsRate(PaymentLeg, DefaultLeg);
}

#endif //PremiaCurrentVersion

```