

[Help](#)

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

/// \ file cdsmkt.cpp
/// \ brief CDS_NoCorr_MarketData 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 "cdsmkt.h"
#include "cstring"

using namespace std;

static void Fatal_err(const char text[100])
{

    char string[100];
    strcpy(string, "*** Error: ");
    strcat(string, text);
    throw logic_error(string);
}

CDS_NoCorr_MarketData::
CDS_NoCorr_MarketData(double Z, vector<double> &timesT,
                      string inputIntensity,
                      string inputZC):
    _Z(Z),
    _timesT(timesT),
    _pConstShortRate(inputZC),
    numInt(this)
{
    ReadData(_pLinIntensity, inputIntensity);
    ReadData(_curveZC, inputZC);
}
```

```

    I1 = 0;
    I2 = 0;
    //cout << "CDS_NoCorr_MarketData, T: " << _timesT[_timesT.size() - 1] << endl;
}

```

```

CDS_NoCorr_MarketData::
CDS_NoCorr_MarketData(vector<double> &intensityMat, vector<double> &intensityRat
                        vector<double> &RatesMat,
                        vector<double> &Rates,
                        double maturity, double period,
                        double recovery):
    _Z(1 - recovery),
    _pConstShortRate(RatesMat, Rates),
    numInt(this)
{
    ReadData(_pLinIntensity, intensityMat, intensityRates);
    ReadData(_curveZC, RatesMat, Rates);
    I1 = 0;
    I2 = 0;

    _timesT.push_back(0.0);
    double t, yearFrac;
    t = yearFrac = (12. / period);
    _periodN = static_cast<int>(maturity / yearFrac);
    for (int i = 0; i < _periodN; i++)
    {
        _timesT.push_back(t);
        t += yearFrac;
    }
}

```

```

void CDS_NoCorr_MarketData::Write(vector<DateRate> &data, string outputFileName)
{
    ofstream out(outputFileName.c_str());
    unsigned int i;
    for (i = 0; i < data.size() - 1; i++)
        out << data[i].date << " " << data[i].rate << "\ n";
    out << data[i].date << " " << data[i].rate;
}

```

```

double CDS_NoCorr_MarketData::f2(double u)

```

```

{
    return MarketZC(u) * ComputeIntensity(u) * exp(-IntegralPLin(u));
}

double CDS_NoCorr_MarketData::f1(double u)
{
    int i = 0;
    double T_beta;
    while (u > _timesT[i + 1])
    {
        i++;
    }
    T_beta = _timesT[i];

    return f2(u) * (u - T_beta);
}

double CDS_NoCorr_MarketData::f_Sum(int n0, int n) const
{
    if (n0 > n)
    {
        Fatal_err("** Error: in the routine CDS_NoCorr_MarketData::f_Sum. Exit.");
    }

    double s = 0;
    int i;
    for (i = n0; i <= n; i++)
    {
        s += MarketZC(_timesT[i]) * (_timesT[i] - _timesT[i - 1])
            * exp(-IntegralPLin(_timesT[i]));
    }

    return s;
}

double CDS_NoCorr_MarketData::ComputeIntensity(double t) const
{
    double x1 = _pLinIntensity[0].date;
    double y1 = _pLinIntensity[0].rate;
    double x2;
    double y2;

```

```

    if (t < x1)
    {
        return 0.0;
    }

    double a, b;
    unsigned int i = 1;
    while ((t > _pLinIntensity[i].date) && (i < _pLinIntensity.size()))
    {
        i++;
    }

    if (i == _pLinIntensity.size())
    {
        return 0;
    }

    if (t == _pLinIntensity[i].date)
        return _pLinIntensity[i].rate;

    x1 = _pLinIntensity[i - 1].date;
    y1 = _pLinIntensity[i - 1].rate;
    x2 = _pLinIntensity[i].date;
    y2 = _pLinIntensity[i].rate;
    a = (y1 - y2) / (x1 - x2);
    b = y1 - x1 * a;
    //segment(x1, y1, x2, y2, &a, &b);

    return a * t + b;
}

double CDS_NoCorr_MarketData::CdsRate(double T, int noTi)
{
    double Ta = 0, Tc;
    int index = 1;

    double I1 = 0., I2 = 0., S;

```

```

do
{
    Tc = _timesT[ index ];

    I1 += numInt.compute(&CDS_NoCorr_MarketData::f1, Ta, Tc);
    I2 += numInt.compute(&CDS_NoCorr_MarketData::f2, Ta, Tc);

    index++;
    Ta = Tc;
}
while (Ta < T);

S = f_Sum(1, noTi);

/*
cout << "I1, I2, S: " << I1 << " " << I2 << " " << S << endl;
cout << "Default Leg: " << _Z*I2 << endl << "Payment Leg: " << I1 + S << endl;
*/

return (_Z * I2) / (I1 + S);
}

double CDS_NoCorr_MarketData::CdsRate(double T, int noTi, double &paymentLeg, do
{
    double Ta = 0, Tc;
    int index = 1;

    double I1 = 0., I2 = 0., S;

    do
    {
        Tc = _timesT[ index ];

        I1 += numInt.compute(&CDS_NoCorr_MarketData::f1, Ta, Tc);
        I2 += numInt.compute(&CDS_NoCorr_MarketData::f2, Ta, Tc);

        index++;
        Ta = Tc;
    }
    while (Ta < T);

```

```

    S = f_Sum(1, noTi);

    defaultLeg = _Z * I2;
    paymentLeg = I1 + S;

    return (_Z * I2) / (I1 + S);
}

double CDS_NoCorr_MarketData::CdsRate(double T, int noTi,
                                       double &I1, double &I2, double &S)
{
    double Ta = 0, Tc;
    int index = 1;

    I1 = 0.;
    I2 = 0.;

    do
    {
        Tc = _timesT[ index ];

        I1 += numInt.compute(&CDS_NoCorr_MarketData::f1, Ta, Tc);
        I2 += numInt.compute(&CDS_NoCorr_MarketData::f2, Ta, Tc);

        index++;
        Ta = Tc;
    }
    while (Ta < T);

    S = f_Sum(1, noTi);

    return (_Z * I2) / (I1 + S);
}

double CDS_NoCorr_MarketData::CDS(double T, int noTi, double Rf)
{
    CdsRate(T, noTi, I1, I2, S);
    return Rf * (I1 + S) - _Z * I2;
}

```

```

//compute Integral_0^t _pLinShortRate(s) ds
double CDS_NoCorr_MarketData::IntegralPLin(double t) const
{
    int dim = _pLinIntensity.size();

    double x1 = _pLinIntensity[0].date;
    double y1 = _pLinIntensity[0].rate;
    double x2;
    double y2;

    if (t <= x1) return 0.0;

    double a, b;
    double sum = 0.0;
    int i = 1;
    while ((t > _pLinIntensity[i].date) && (i < dim))
    {
        x2 = _pLinIntensity[i].date;
        y2 = _pLinIntensity[i].rate;
        a = (y1 - y2) / (x1 - x2);
        b = y1 - x1 * a;

        sum += (a * (x2 * x2 - x1 * x1)) / 2. + b * (x2 - x1);

        x1 = x2;
        y1 = y2;
        i++;
    }
    if (i == dim) return sum;

    x2 = _pLinIntensity[i].date;
    y2 = _pLinIntensity[i].rate;
    a = (y1 - y2) / (x1 - x2);
    b = y1 - x1 * a;

    sum += (a * (t * t - x1 * x1)) / 2. + b * (t - x1);
    return sum;
}

```

```

void CDS_NoCorr_MarketData::ReadData(vector<DateRate> &data, string fileName)
{
    ifstream input(fileName.c_str());
    if (!input)
    {
        string s("I Error: no file named ");
        s = s + fileName.c_str();
        Fatal_err(s.c_str());
    }
    ifstream in(fileName.c_str());
    if (in.eof())
    {
        string s("I Error: no data in input file named ");
        s = s + fileName.c_str();
        Fatal_err(s.c_str());
    }

    {
        double date, price;
        in >> date >> price;
        DateRate dp(date, price);
        data.push_back(dp);
    }
    while (!in.eof())
    {
        double date, price;
        in >> date >> price;

        double anteriorDate = data[data.size() - 1].date;
        if (date <= anteriorDate)
        {
            cout << fileName.c_str() << ": aici: " << date << " " << anteriorDate
                << endl;
            Fatal_err("*** Error: Market zero-coupon curve is corrupted!");
        }

        DateRate dp(date, price);
        data.push_back(dp);
    }
}

```



```

}
```

```

void CDS_NoCorr_MarketData::ReadData(vector<DateRate> &curveZC, vector<double> &
{
    if (zcMat.size() != zcRates.size())
    {
        throw logic_error("*** Error: CIRppSR: zcMat and zcRates arrays have not t
    }

    DateRate dp(zcMat[0], zcRates[0]);
    curveZC.push_back(dp);

    for (int i = 1; i < (int)zcMat.size(); i++)
    {
        if (zcMat[i] <= (int)zcMat[i - 1])
        {
            throw logic_error("*** Error: CDS_NoCorr_MarketData: input curve is co
        }
        DateRate dp(zcMat[i], zcRates[i]);
        curveZC.push_back(dp);
    }
}

void PConstShortRate::ReadData(vector<double> &zcMat, vector<double> &zcRates)
{
    if (zcMat.size() != zcRates.size())
    {
        throw logic_error("*** Error: CIRppSR: zcMat and zcRates arrays have not t
    }

    DateCreal dp(zcMat[0], zcRates[0]);
    _curveZC.push_back(dp);

    for (int i = 1; i < (int)zcMat.size(); i++)
    {
        if (zcMat[i] <= zcMat[i - 1])
        {
            throw logic_error("*** Error: CDS_NoCorr_MarketData: input curve is co
```

```

    }
    DateCreal dp(zcMat[i], zcRates[i]);
    _curveZC.push_back(dp);
}
}

```

```

void PConstShortRate::ReadData(string fileName)
{
    ifstream input(fileName.c_str());
    if (!input)
    {
        string s("I Error: no file named ");
        s = s + fileName.c_str();
        Fatal_err(s.c_str());
    }
    ifstream in(fileName.c_str());
    if (in.eof())
    {
        string s("I Error: no data in input file named ");
        s = s + fileName.c_str();
        Fatal_err(s.c_str());
    }

    {
        double date, price;
        in >> date >> price;
        DateCreal dp(date, price);
        _curveZC.push_back(dp);
    }
    while (!in.eof())
    {
        double date, price;
        in >> date >> price;

        double anteriorDate = _curveZC[_curveZC.size() - 1].date;
        if (date <= anteriorDate)
        {
            Fatal_err("*** Error: Market zero-coupon curve is corrupted!");
        }
    }
}

```

[illegible]

```

ReadData(RatesMat, Rates);

_dim = _curveZC.size();
if (_dim < 2)
    Fatal_err("Insufficient data!");

double r1 = -log(_curveZC[1].r) / _curveZC[1].date;
DateCreal dr0(_curveZC[1].date, r1);
_pConstShortRate.push_back(dr0);

for (int i = 2; i < _dim; i++)
{
    double P_Tim1 = _curveZC[i - 1].r;
    double Tim1 = _curveZC[i - 1].date;
    double P_Ti = _curveZC[i].r;
    //cout << _curveZC[i].r << " " << _curveZC[i-1].r << endl;
    double Ti = _curveZC[i].date;
    double r_i = (-log(P_Ti / P_Tim1)) / (Ti - Tim1);

    DateCreal dr(Ti, r_i);
    _pConstShortRate.push_back(dr);
}

}

double PConstShortRate::f0_t(double t) const
{
    int i = 1;
    while ((t > _pConstShortRate[i - 1].date) && (i < _dim))
        i++;

    if (i > _dim)
    {
        return 0;
    }
    return _pConstShortRate[i - 1].r;
}

double PConstShortRate::ComputeShortRate(double t) const
{
    int i = 1;

```

```

double f0_t = 0.0;
while ((t >= _pConstShortRate[i - 1].date) && (i < _dim))
{
    //cout << i << " " << _pConstShortRate[i-1].date << endl;
    f0_t += _pConstShortRate[i - 1].r * (_curveZC[i].date - _curveZC[i - 1].date);
    i++;
}

if (i > _dim)
{
    return 0;
}

if (t == _pConstShortRate[i].date)
    return f0_t;

f0_t += _pConstShortRate[i - 1].r * (t - _curveZC[i - 1].date);

return f0_t;
}

double PConstShortRate::ComputeZC(double t) const
{
    return exp(-ComputeShortRate(t));
}

#endif //PremiaCurrentVersion

```