

Help

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

/// \ file gm.cpp
/// \ brief gaussian mapping technique
/// \ 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 <iostream>
#include <fstream>
#include <iomanip>
#include <cstring>
#include <
href../../common/math/highdim_solver/highdim_vector_h_src.pdfvector>
#include <cassert>
#include <cmath>

#include "
href../../common/math/credit_cds/gm_h_src.pdfgm.h"

using namespace std;

double g(double k, double T)
{
    return (1 - exp(-k * T)) / k;
}

static void Fatal_err(const char text[100])
{
    char string[100];
    strcpy(string, "*** Error: ");
    strcat(string, text);
    throw logic_error(string);
}
```

```
}
```

```
CIR_Mapped_toVasicek::
```

```
CIR_Mapped_toVasicek(double k, double theta, double sigma, double x0, double T):
```

```
    _k(k), _theta(theta), _sigma(sigma), _x0(x0), _T(T)
```

```
{
```

```
    _h = sqrt(SQR(_k) + 2 * SQR(_sigma));
```

```
    _sigma_v = Compute_VasicekMappedVolatility(T);
```

```
}
```

```
CDS_GaussianMapping_Old::
```

```
CDS_GaussianMapping_Old(double k, double theta, double sigma, double x0,
```

```
                        string inputCDS,
```

```
                        double k_r, double theta_r, double sigma_r, double x0_r,
```

```
                        string inputShortRate,
```

```
                        double rho,
```

```
                        vector<double> &timesT, double Z):
```

```
    _k(k), _theta(theta), _sigma(sigma), _x0(x0),
```

```
    _k_r(k_r), _theta_r(theta_r), _sigma_r(sigma_r), _x0_r(x0_r),
```

```
    _timesT(timesT),
```

```
    _Z(Z),
```

```
    _cirpp_intensity(k, theta, sigma, x0, timesT[timesT.size() - 1], inputCDS),
```

```
    _cirpp_shortRate(k_r, theta_r, sigma_r, x0_r,
```

```
                    timesT[timesT.size() - 1], inputShortRate),
```

```
    _shortRate(k_r, theta_r, sigma_r, x0_r, timesT[timesT.size() - 1]),
```

```
    _intensity(k, theta, sigma, x0, timesT[timesT.size() - 1]),
```

```
    _rho(rho),
```

```
    numInt(this)
```

```
{
```

```
    //cout << "CDS_GaussianMapping_Old...\ n";
```

```
    if (rho < -1 || rho > 1)
```

```
        Fatal_err("CDS_GaussianMapping_Old: Correlation must be in [-1, 1].");
```

```
    I1 = 0;
```

```
    I2 = 0;
```

```
    _T = _timesT[_timesT.size() - 1];
```

```
    //numInt.setPt(this);
```

```
}
```

```
CDS_GaussianMapping_Old::
```

```
CDS_GaussianMapping_Old(double k, double theta, double sigma, double x0,
```

```

        vector<double> &intensityMat,
        vector<double> &intensityRates,
        double k_r, double theta_r, double sigma_r, double x0_r,
        vector<double> &RatesMat,
        vector<double> &Rates,
        double rho,
        vector<double> &timesT, double Z):
_k(k), _theta(theta), _sigma(sigma), _x0(x0),
_k_r(k_r), _theta_r(theta_r), _sigma_r(sigma_r), _x0_r(x0_r),

_timesT(timesT),
_Z(Z),
_cirpp_intensity(k, theta, sigma, x0, timesT[timesT.size() - 1], intensityMat,
_cirpp_shortRate(k_r, theta_r, sigma_r, x0_r,
                 timesT[timesT.size() - 1], RatesMat, Rates),
_shortRate(k_r, theta_r, sigma_r, x0_r, timesT[timesT.size() - 1]),
_intensity(k, theta, sigma, x0, timesT[timesT.size() - 1]),
_rho(rho),
numInt(this)
{
    //cout << "CDS_GaussianMapping_Old...\ n";

    if (rho < -1 || rho > 1)
        Fatal_err("CDS_GaussianMapping_Old: Correlation must be in [-1, 1].");

    I1 = 0;
    I2 = 0;
    _T = _timesT[_timesT.size() - 1];
    //numInt.setPt(this);
}

double CDS_GaussianMapping_Old::Compute_M2_Vasicek(double T)
{
    if (T == 0) return 1;
    return exp(-mean_A(T) + 0.5 * variance_A(T));
}

double CDS_GaussianMapping_Old::Compute_M2_Vasicek_Deg(double T)
{
    return exp(-mean_Ad(T) + 0.5 * variance_Ad(T));
}

```

```

double CDS_GaussianMapping_Old::Compute_M1_Vasicek(double t)
{
    if (t == 0) return _x0;
    double term = rho_bar(t) * sqrt(variance_A(t) * variance_B(t));
    double in_exp = -mean_A(t) + 0.5 * (1 - SQR(rho_bar(t))) * variance_A(t);

    return mean_B(t) * Compute_M2_Vasicek(t) - term * exp(in_exp);
}

double CDS_GaussianMapping_Old::Compute_M1_Vasicek_Deg(double t)
{
    if (t == 0) return _x0;
    double term = rho_bard(t) * sqrt(variance_Ad(t) * variance_B(t));
    double in_exp = -mean_Ad(t) + 0.5 * (1 - SQR(rho_bard(t))) * variance_Ad(t);

    return mean_B(t) * Compute_M2_Vasicek_Deg(t) - term * exp(in_exp);
}

double CDS_GaussianMapping_Old::Compute_M1_Vasicek_Corrected(double t)
{
    if (t == 0) return 1;
    //cout << " Compute_M1_Vasicek: " << Compute_M1_Vasicek(t) << " Delta: " << D

    return Compute_M1_Vasicek(t) + Delta(t);
}

double CDS_GaussianMapping_Old::Compute_M1_Vasicek_Corrected_num(double t)
{
    //cout <<Compute_Mean1_Vasicek(t) << " " << Delta(t) << endl;
    return Compute_M1_Vasicek(t) + Delta_num(t);
}

double CDS_GaussianMapping_Old::mean_A(double t)
{
    return (_theta + _theta_r) * t
        - (_theta - _x0) * g(_k, t) - (_theta_r - _x0_r) * g(_k_r, t);
}

```

```
double CDS_GaussianMapping_Old::mean_Ad(double t)
{
    return _theta * t - (_theta - _x0) * g(_k, t);
}
```

```
double CDS_GaussianMapping_Old::mean_B(double t)
{
    //
    return _theta - (_theta - _x0) * exp(-_k * t);
}
```

```
double CDS_GaussianMapping_Old::variance_A(double t)
{
    double term = SQR(sigma_v(t) / _k) * (t - 2 * g(_k, t) + g(2 * _k, t));
    double term_r = SQR(sigma_v_r(t) / _k_r) * (t - 2 * g(_k_r, t) + g(2 * _k_r, t));
    double middle =
        ((2 * _rho * sigma_v(t) * sigma_v_r(t)) / (_k * _k_r))
        * (t - g(_k, t) - g(_k_r, t) + g(_k + _k_r, t));

    return term + middle + term_r;
}
```

```
double CDS_GaussianMapping_Old::variance_Ad(double t)
{
    double term = SQR(sigma_v(t) / _k) * (t - 2 * g(_k, t) + g(2 * _k, t));
    return term;
}
```

```
double CDS_GaussianMapping_Old::variance_B(double t)
{
    return SQR(sigma_v(t)) * g(2 * _k, t);
}
```

```
double CDS_GaussianMapping_Old::rho_bar(double t)
{
    double sigma_A = sqrt(variance_A(t));
    double sigma_B = sqrt(variance_B(t));
```

```

//cout << "rho_bar: " << t << " " << variance_A(t) << " " << variance_B(t)<<
if (sigma_A == 0 || sigma_B == 0)
    Fatal_err("CDS_GaussianMapping_Old::rho_bar: division by 0.");

double term1 = (SQR(sigma_v(t)) / _k) * (g(_k, t) - g(2 * _k, t));
double term2 = (_rho * sigma_v(t) * sigma_v_r(t) / _k_r) * (g(_k, t) - g(_k_r

//cout << "rho_bar: " << term1 << " " << term2 << " " << sigma_A << " " << s

return (term1 + term2) / (sigma_A * sigma_B);
}

double CDS_GaussianMapping_Old::rho_bard(double t)
{
    double var_Ad = variance_Ad(t);
    assert(var_Ad >= 0);

    double sigma_Ad = sqrt(var_Ad);
    double sigma_B = sqrt(variance_B(t));

    if (sigma_Ad == 0 || sigma_B == 0)
        Fatal_err("CDS_GaussianMapping_Old::rho_bar: division by 0.");

    double term1 = (SQR(sigma_v(t)) / _k) * (g(_k, t) - g(2 * _k, t));

    return term1 / (sigma_Ad * sigma_B);
}

double CDS_GaussianMapping_Old::Delta(double t)
{
    if (t == 0) return 0;

    double zc_cir =
        _shortRate.Compute_ZC_CIRn(t) * _intensity.Compute_ZC_CIR_d(t);
    double zc_vasi =
        _shortRate.Compute_ZC_Vasicekn(t) * Compute_M1_Vasicek_Deg(t);
    return zc_cir - zc_vasi;
}

```

```

double CDS_GaussianMapping_Old::Delta_num(double t)
{
    if (t == 0) return 0;

    double zc_cir =
        -_shortRate.Compute_ZC_CIRn(t) * _intensity.Compute_ZC_CIR_d_num(t);
    double zc_vasi = _shortRate.Compute_ZC_Vasicekn(t) * Compute_M1_Vasicek_Deg(t);
    //double zc_vasi = _shortRate.Compute_ZC_Vasicekn(t) * _intensity.Compute_ZC_Va
    return zc_cir - zc_vasi;
}

double CIR_Mapped_toVasicek::Compute_ZC_CIRn(double t)
{
    return A_CIR(t) * exp(-B_CIR(t) * _x0);
}

double CIR_Mapped_toVasicek::H_CIR(double t)
{
    return H_CIR_n(t) / B_CIR_d(t);
    //return (_h*exp((_h+_k)*t*0.5)) / B_CIR_d(t);
}

double CIR_Mapped_toVasicek::H_CIR_d(double t)
{
    return (H_CIR_n_d(t) * B_CIR_d(t) - B_CIR_d_d(t) * H_CIR_n(t)) / SQR(B_CIR_d(t)
}

double CIR_Mapped_toVasicek::H_CIR_n(double t)
{
    return _h * exp((_h + _k) * t * 0.5);
}

double CIR_Mapped_toVasicek::H_CIR_n_d(double t)
{
    return _h * ((_h + _k) * 0.5) * exp((_h + _k) * t * 0.5);
}

double CIR_Mapped_toVasicek::A_CIR(double t)
{

```

```

    return pow(H_CIR(t), (2 * _k * _theta) / SQR(_sigma));
}
double CIR_Mapped_toVasicek::A_CIR_d(double t)
{
    double _power = (2 * _k * _theta) / SQR(_sigma);
    return _power * pow(H_CIR(t), _power - 1) * H_CIR_d(t);
}

double CIR_Mapped_toVasicek::B_CIR_n(double t)
{
    return exp(t * _h) - 1;
}
double CIR_Mapped_toVasicek::B_CIR_n_d(double t)
{
    return _h * exp(t * _h);
}

double CIR_Mapped_toVasicek::B_CIR_d(double t)
{
    return _h + 0.5 * (_k + _h) * B_CIR_n(t);
}
double CIR_Mapped_toVasicek::B_CIR_d_d(double t)
{
    return _h * 0.5 * (_k + _h) * exp(t * _h);
}

double CIR_Mapped_toVasicek::B_CIR(double t)
{
    return B_CIR_n(t) / B_CIR_d(t);
}
double CIR_Mapped_toVasicek::B_CIR_deriv(double t)
{
    return
        (B_CIR_n_d(t) * B_CIR_d(t) - B_CIR_d_d(t) * B_CIR_n(t)) / SQR(B_CIR_d(t));
}

double CIR_Mapped_toVasicek::Compute_ZC_CIR(double t)
{
    if (t < 0.0001)

```



```

    return 1;
    double h = sqrt(SQR(_k) + 2 * SQR(_sigma));
    double exp_th = exp(t * h);
    double denominator = h + 0.5 * (_k + h) * (exp_th - 1);
    double A = (h * exp((h + _k) * t * 0.5)) / denominator;
    A = pow(A, (2 * _k * _theta) / SQR(_sigma));
    double B = (exp_th - 1) / denominator;

    return A * exp(-B * _x0);
}

double CIR_Mapped_toVasicek::Compute_ZC_CIR_d(double t)
{
    double deriv1 = A_CIR_d(t) * exp(-B_CIR(t) * _x0);
    double deriv2 = -A_CIR(t) * B_CIR_deriv(t) * _x0 * exp(-B_CIR(t) * _x0);
    return deriv1 + deriv2;
}

// :WRONG:
double CIR_Mapped_toVasicek::Compute_ZC_CIR_d_num(double t)
{
    //numerical differenciation: a three-point fomula
    //burden & faires, numerical analysis, pg 161
    double deriv;

    if (t - 0.5 * INC > 0)
    {
        deriv = (Compute_ZC_CIRn(t + 0.5 * INC) - Compute_ZC_CIRn(t - 0.5 * INC))
    }

    else
        // :WRONG:
        deriv = Compute_ZC_CIR(INC) / INC;
    return deriv;
}

double CIR_Mapped_toVasicek::Compute_VasicekMappedVolatility(double T)
{
    double numerator =

```

```

    log(Compute_ZC_CIR(T)) + _theta * T - (_theta - _x0) * g(_k, T);

double denominator = T - 2 * g(_k, T) + g(2 * _k, T) ;
// :PROBLEM::WARNING:
// For small values of the difference _T-_t the equation has no solution:
// there is no Vasicek mapped volatility in (0, 1).
// Theoretically, the solution always exists, but numerically this is not
// the case.
//
// For small values of the difference _T-_t the numerator and the
// denominator become negative, and the quotient numerator/denominator
// becomes negative; thus the sqrt(quotient) returns NaN; the choice of
// the value that I return in this case is justified in the paper of
// Brigo and Alfonsi.
//
// The closed-form expression is not stable for small differences _T-_t
// This problem is due to numerical errors made by the computer !
//
// A more clever solution consists in applying a modified bisection
// method. This searches the solution in the interval [0, 1], but never
// evaluates the objective function in 0, which can be justified by
// theoretical arguments.
// This would be a stable solution.

if ((numerator < 0) || (denominator < 0))
    return _sigma * sqrt(_x0);

return _k * sqrt(2 * (numerator / denominator));
}

double CIR_Mapped_toVasicek::Compute_ZC_Vasicek(double t)
{
    //double sigma_v = Compute_VasicekMappedVolatility(t);
    //cout << "v : " << sigma_v << endl; cout << "_v: " << _sigma_v << endl;
    double term1 = -_theta * t + (_theta - _x0) * g(_k, t);
    double term2 = 0.5 * SQR(_sigma_v / _k) * (t - 2 * g(_k, t) + g(2 * _k, t));
    return exp(term1 + term2);
}

double CIR_Mapped_toVasicek::A_Vasicek(double t)
{

```

```

    double term1 = -_theta * t + (_theta - _x0) * g(_k, t);
    double term2 = 0.5 * SQR(_sigma_v / _k) * (t - 2 * g(_k, t) + g(2 * _k, t));
    return term1 + term2;
}

double CIR_Mapped_toVasicek::Compute_ZC_Vasicekn(double t)
{
    return exp(A_Vasicek(t));
}

void CDS_GaussianMapping_Old::Get_sigmas(double t)
{
    cout << "sigma_v intensity, in " << t << ": " << sigma_v(t) << endl;
    cout << "sigma_v short-rate, in " << t << ": " << sigma_v_r(t) << endl;
}

double CDS_GaussianMapping_Old::f2(double u)
{
    //cout << u << " " << _r << " " << MarketZC(u) << " " << ComputeIntensity(u) << endl;
    //int j;cin >> j;

    //double _exp_ = exp( -NumericalIntegration(&CDS_GaussianMapping::sum_of_shift

    double _exp_intensity = exp(-_cirpp_intensity.NumericalIntegration_ofPhi_SS(u));
    double _exp_shortRate = exp(-_cirpp_shortRate.NumericalIntegration_ofPhi_SS(u));
    //double _exp_shortRate = _cirpp_shortRate.NumericalIntegration_ofExpPhi(u);
    double _mean1 = Compute_M1_Vasicek_Corrected(u);
    double _mean2 = Compute_M2_Vasicek(u);

    /*
    cout << " _exp_intensity: " << _exp_intensity;
    cout << " _exp_shortRate: " << _exp_shortRate;
    cout << " _mean1: " << _mean1;
    cout << " _mean2: " << _mean2;
    */

    return _exp_intensity * _exp_shortRate * (_mean1 + _cirpp_intensity.Phi(u) * _
}

```

```

double CDS_GaussianMapping_Old::f1(double u)
{
    int i = 0;
    double T_beta_minus_1;
    while (u > _timesT[i + 1])
    {
        i++;
    }
    T_beta_minus_1 = _timesT[i];
    //cout << "f2: " << f2(u);

    return f2(u) * (u - T_beta_minus_1);
}

double CDS_GaussianMapping_Old::f_Sum(int n0, int n)
{
    if (n0 > n)
    {
        Fatal_err("** Error: in the routine CDS_GaussianMapping::f_Sum. Bad input")
    }

    double s = 0;
    //cout << n0 << " " << n << endl;
    int i;
    for (i = n0; i <= n; i++)
    {
        //double _exp_ = exp( -NumericalIntegration(&CDS_GaussianMapping::sum_of_s

        double _exp_intensity = exp(-_cirpp_intensity.NumericalIntegration_ofPhi_S
        double _exp_shortRate = exp(-_cirpp_shortRate.NumericalIntegration_ofPhi_S
        double _mean2 = Compute_M2_Vasicek(_timesT[i]);
        s += (_timesT[i] - _timesT[i - 1]) * _exp_intensity * _exp_shortRate * _me

        /*
        cout << i << " " << _timesT[i];
        cout << " exp(-Int Phi_int): " << " " << _exp_intensity << " exp(-Int Phi_
        */
        //cout << i << " " << _timesT[i] << " M2: " << _exp_intensity * _exp_short

```

```

    }

    return s;
}

double CDS_GaussianMapping_Old::Quote(double T, int noTi, double &defaultLeg, double &paymentLeg)
{
    double Ta = _timesT[0], Tc;
    int index = 1;

    do
    {
        Tc = _timesT[ index ];

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

#endif //PremiaCurrentVersion

```