

Help

```
#include "lmm1d_std.h"

#include "math/lmm/lmm_libor.h"
#include "math/lmm/lmm_products.h"
#include "math/lmm/lmm_volatility.h"
#include "math/lmm/lmm_numerical.h"
#include "math/lmm/lmm_zero_bond.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
static int CHK_OPT(AP_Swaption_LMM)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_Swaption_LMM)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

// Analytical approximation formula for the Swaption Black Volatility
// Consider a swaption with first reset date T(alpha), n payment dates T(alpha+
// The approximate formula is :
// vol_swaption = sum for (i=0:n-1 and j=0:n-1) of w(i)*w(j)*L(i)*L(j)* integral
// w(i) and swap_rate(0) are function of the libor rates (L(k))k=0:n-1, sigma(t,
// Rmk : L(k) = L(Tk,Tk,Tk+1) : value at date Tk of the libor rate set at Tk pay
// swap_maturity = Tn, swaption_maturity = To, tenor = Tk+1 - Tk
static void ap_swaption_black_volatility(Libor *ptLib, Volatility *ptVol, double
{
    int i, j, k, l, alpha, beta, n, Nfac, Nstep_integration;
    double t, vol_swaption, integrale, somme_integrale, Ti, Tj, dt;
    PnlVect *weight;
    PnlVect *zc;

    weight = pnl_vect_create(0);
    zc = pnl_vect_create(0);

    alpha = (int)(swaption_maturity / tenor); // index of swaption_maturity
    beta = (int)(swap_maturity / tenor); // index of swap_maturity
    n = beta - alpha; // Nbr of payments dates
```

```

Nfac = ptVol->numberOfFactors; // Nbr of factors in diffusion process

Libor_To_ZeroCoupon(ptLib, zc); // Compute ZeroCoupon bond from Libor vector
(*weight) = pnl_vect_wrap_subvect(zc, alpha + 1, n); // extract the zc bond P(

*sum_discount_factor = pnl_vect_sum(weight);

pnl_vect_div_double(weight, *sum_discount_factor); // Normilization of the wei

*sum_discount_factor *= tenor;

// swap_rate(0) = sum over i of weight(i)*LiborRate(0,Ti,Ti+1) , see Brigo&Mer
*swap_rate = 0;
for (i = 0; i < n ; i++)
{
    *swap_rate += GET(weight, i) * GET(ptLib->libor, alpha + i);
}

Nstep_integration = 40; // number of step used to compute the integral of vola
dt = (swaption_maturity - valuation_date) / Nstep_integration; // step for the
vol_swaption = 0; // Black's volatility of the swaption

for (i = 0; i < n ; i++)
{
    Ti = swaption_maturity + i * tenor;
    for (j = 0; j < n ; j++)
    {
        Tj = swaption_maturity + j * tenor;
        somme_integrale = 0;
        for (k = 0; k < Nfac; k++) // computation of the integral of volatilit
        {
            // We use the simple trapezoidal rule
            integrale = evalVolatility(ptVol, k, valuation_date, Ti) * evalVol
            integrale += evalVolatility(ptVol, k, swaption_maturity, Ti) * eva
            integrale *= 0.5;

            for (l = 1 ; l < Nstep_integration; l++)
            {
                t = valuation_date + l * dt;
                integrale += evalVolatility(ptVol, k, t, Ti) * evalVolatility(
            }
        }
    }
}

```

```

        integrale *= dt;
        somme_integrale += integrale;
    }

    vol_swaption += GET(weight, i) * GET(weight, j) * GET(ptLib->libor, d1);
}

vol_swaption = vol_swaption / SQR(*swap_rate) ;

*black_volatility = sqrt(vol_swaption);

pnl_vect_free(&weight);
pnl_vect_free(&z);
}

static int lmm_swaption(NumFunc_1 *p, double l0, double sigma, int nb_factors, double d1, double d2, double black_volatility, double swap_rate, double sum_discount_factor, Volatility *ptVol, Libor *ptLib)
{
    int Nbr_Maturities, payer_or_receiver;
    double d1, d2, black_volatility, swap_rate, sum_discount_factor;

    Volatility *ptVol;
    Libor *ptLib;

    black_volatility = 0;
    swap_rate = 0;
    sum_discount_factor = 0;

    Nbr_Maturities = (int)(swap_maturity / tenor);

    mallocLibor(&ptLib , Nbr_Maturities, tenor, l0);

    mallocVolatility(&ptVol, nb_factors, sigma);

    payer_or_receiver = ((p->Compute) == &Put);

    // Computation of the Swaption Black Volatility
    ap_swaption_black_volatility(ptLib, ptVol, 0., swap_maturity, swaption_maturity, swap_rate, sum_discount_factor, &black_volatility);

    d1 = (log(swap_rate / swaption_strike)) / black_volatility + 0.5 * black_volatility * swap_maturity;

```

```

    d2 = d1 - black_volatility;

    if (payer_or_receiver == 1) // Case of Payer Swaption
    {
        *price = Nominal * sum_discount_factor * (swap_rate * cdf_nor(d1) - swapti
    }

    else if (payer_or_receiver == 0) // Case of Receiver Swaption
    {
        *price = Nominal * sum_discount_factor * (swap_rate * (cdf_nor(d1) - 1) -
    }

    freeLibor(&ptLib);
    freeVolatility(&ptVol);

    return (OK);
}

int CALC(AP_Swaption_LMM)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return lmm_swaption(ptOpt->PayOff.Val.V_NUMFUNC_1,
                        ptMod->l0.Val.V_PDOUBLE,
                        ptMod->Sigma.Val.V_PDOUBLE,
                        ptMod->NbFactors.Val.V_ENUM.value,
                        ptOpt->Nominal.Val.V_PDOUBLE,
                        ptOpt->ResetPeriod.Val.V_DATE,
                        ptOpt->OMaturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                        ptOpt->BMaturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                        ptOpt->FixedRate.Val.V_PDOUBLE,
                        &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(AP_Swaption_LMM)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PayerSwaption") == 0) || (strcmp(((Option
        return OK;
    else

```

```
        return WRONG;
    }
#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "ap_rebonato_swaption";
    }

    return OK;
}

PricingMethod MET(AP_Swaption_LMM) =
{
    "AP_Swaption_LMM",
    {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_Swaption_LMM),
    {"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(AP_Swaption_LMM),
    CHK_ok,
    MET(Init)
} ;
```