

Help

```
#include "config.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_root.h"

/*****
/* Written and (C) by David Pommier <pommier.david@gmail.com> */
/* 2008 */
/*
/* This program is free software; you can redistribute it and/or modify */
/* it under the terms of the GNU General Public License as published by */
/* the Free Software Foundation; either version 3 of the License, or */
/* (at your option) any later version. */
/*
/* This program is distributed in the hope that it will be useful, */
/* but WITHOUT ANY WARRANTY; without even the implied warranty of */
/* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the */
/* GNU General Public License for more details. */
/*
/* You should have received a copy of the GNU General Public License */
/* along with this program. If not, see <http://www.gnu.org/licenses/>. */
*****/

/**
 * \ defgroup Pnl_Data_Vol_Impli_BS Implied Volatlity
 */
/*@{*/
typedef struct Pnl_Data_Vol_Impli_BS
{
    int is_call;
    double Price, Bond, Forward, Strike, Maturity;
} Pnl_Data_Vol_Impli_BS;
/*@}*/

/**
 * Price a forward contract
```

```

*
* @param Spot current value.
* @param r interest rate value
* @param divid value of dividend
* @param Maturity a double, echeance time (T)
* @return price of a forward contract
* Forward = Spot*exp((r-divid)*Maturity) a double,
* price of Spot at time T - maturity
*/
double pnl_forward_price(double Spot, double r, double divid, double Maturity)
{
    return Spot * exp((r - divid) * Maturity);
}

/**
* Price a call in BS model
*
* @param Vol a double, the volatility
* @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond
* @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
* time T - maturity.
* @param Strike a double, for value contract
* @param Maturity a double, echeance time (T)
* @return price of a call option (pay max(S-K,0) at time T
*/
double pnl_bs_impli_call(double Vol, double Bond, double Forward, double Strike,
{
    double V_Sqrt_T;
    double D1;
    double D2;

    PNL_CHECK(Vol < 0., "Volatility required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Maturity < 0., "Maturity required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Forward <= 0., "Maturity required to be > 0", "pnl_bs_impli_put");
    PNL_CHECK(Strike < 0., "Strike required to be >= 0", "pnl_bs_impli_put");

    if (Strike == 0.) return Bond * Forward;
    if (Vol == 0.0 || Maturity == 0.0) return Bond * MAX(Forward - Strike, 0.0);
    V_Sqrt_T = Vol * sqrt(Maturity);
    D1 = 0.5 * V_Sqrt_T - log(Strike / Forward) / (V_Sqrt_T);
    D2 = D1 - V_Sqrt_T;

```

```

    return Bond * (Forward * cdf_nor(D1) - Strike * cdf_nor(D2));
}

/**
 * Price a put in BS model
 *
 * @param Vol a double, the volatility
 * @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond
 * @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
 * time T - maturity.
 * @param Strike a double, for value contract
 * @param Maturity a double, echeance time (T)
 * @return price of a put option (pay max(K-S,0) at time T
 */
double pnl_bs_impli_put(double Vol, double Bond, double Forward, double Strike,
{
    double V_Sqrt_T;
    double PD1;
    double PD2;

    PNL_CHECK(Vol < 0., "Volatility required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Maturity < 0., "Maturity required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Forward <= 0., "Maturity required to be > 0", "pnl_bs_impli_put");
    PNL_CHECK(Strike < 0., "Strike required to be >= 0", "pnl_bs_impli_put");

    if (Strike == 0.) return 0.;
    if (Maturity == 0.0 || Vol == 0.) return Bond * MAX(Strike - Forward, 0.0);
    V_Sqrt_T = Vol * sqrt(Maturity);
    PD1 = log(Strike / Forward) / (V_Sqrt_T) - 0.5 * V_Sqrt_T;
    PD2 = PD1 + V_Sqrt_T;
    return Bond * (Strike * cdf_nor(PD2) - Forward * cdf_nor(PD1));
}

/**
 * give the delta forward of a call option in BS model
 * note :
 * Delta_Forward exp((r-divid)*Maturity) = Delta
 *
 * @param Vol a double, the volatility
 * @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond

```

```

* @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
* time T - maturity.
* @param Strike a double, for value contract
* @param Maturity a double, echeance time (T)
* @return delta of a call option
*/
double pnl_bs_impli_call_delta_forward(double Vol, double Bond, double Forward,
{
    double V_Sqrt_T;
    double D1;

    PNL_CHECK(Vol < 0., "Volatility required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Maturity < 0., "Maturity required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Forward <= 0., "Maturity required to be > 0", "pnl_bs_impli_put");
    PNL_CHECK(Strike < 0., "Strike required to be >= 0", "pnl_bs_impli_put");

    if (Strike == 0.) return Bond;
    if (Vol == 0.0 || Maturity == 0.0) return (Forward > Strike) ? Bond : 0.;
    V_Sqrt_T = Vol * sqrt(Maturity);
    D1 = 0.5 * V_Sqrt_T - log(Strike / Forward) / (V_Sqrt_T);
    return Bond * cdf_nor(D1);
}

/**
* give the delta forward of a put option in BS model
* note :
* Delta_Forward exp((r-divid)*Maturity) = Delta
*
* @param Vol a double, the volatility
* @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond
* @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
* time T - maturity.
* @param Strike a double, for value contract
* @param Maturity a double, echeance time (T)
* @return delta of a put option
*/
double pnl_bs_impli_put_delta_forward(double Vol, double Bond, double Forward, d
{
    double Sqrt_T;
    double PD1;

```

```

PNL_CHECK(Vol < 0., "Volatility required to be >= 0", "pnl_bs_impli_put");
PNL_CHECK(Maturity < 0., "Maturity required to be >= 0", "pnl_bs_impli_put");
PNL_CHECK(Forward <= 0., "Maturity required to be > 0", "pnl_bs_impli_put");
PNL_CHECK(Strike < 0., "Strike required to be >= 0", "pnl_bs_impli_put");

if (Strike == 0.) return 0.;
if (Vol == 0.0 || Maturity == 0.0) return (Strike > Forward) ? -1.*Bond : 0.;
Sqrt_T = sqrt(Maturity);
PD1 = log(Strike / Forward) / (Vol * Sqrt_T) - 0.5 * Vol * Sqrt_T;
return -Bond * cdf_nor(PD1);
}

/**
 * give the price of a call/put option in BS model
 *
 * @param is_call a int, the option type, 1 for call, 0 for put
 * @param Vol a double, the volatility
 * @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond
 * @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
 * time T - maturity.
 * @param Strike a double, for value contract
 * @param Maturity a double, echeance time (T)
 * @return price of a call/put option
 */
double pnl_bs_impli_call_put(int is_call, double Vol, double Bond, double Forward
{
    return is_call ? pnl_bs_impli_call(Vol, Bond, Forward, Strike, Maturity) : pnl

}

/**
 * give the delta forward of a call/put option in BS model
 * note :
 * Delta_Forward exp((r-divid)*Maturity) = Delta
 *
 * @param is_call a int, the option type, 1 for call, 0 for put
 * @param Vol a double, the volatility
 * @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond
 * @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
 * time T - maturity.

```

```

* @param Strike a double, for value contract
* @param Maturity a double, echeance time (T)
* @return delta of a call/put option
*/
double pnl_bs_impli_call_put_delta_forward(int is_call, double Vol, double Bond,
{
    return is_call ? pnl_bs_impli_call_delta_forward(Vol, Bond, Forward, Strike, M
}

/**
* give the first derivative of the price w.r.t the volatility of a call/put
* option in a BS model
*
* @param Vol a double, the volatility
* @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond
* @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
* time T - maturity.
* @param Strike a double, for value contract
* @param Maturity a double, echeance time (T)
* @return vega of a call/put option
*/
double pnl_bs_impli_vega(double Vol, double Bond, double Forward, double Strike,
{
    double V_Sqrt_T;
    double D;

    PNL_CHECK(Vol < 0., "Volatility required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Maturity < 0., "Maturity required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Forward <= 0., "Maturity required to be > 0", "pnl_bs_impli_put");
    PNL_CHECK(Strike < 0., "Strike required to be >= 0", "pnl_bs_impli_put");

    if (Vol == 0.0 || Maturity == 0.0 || Strike == 0.0) return 0.0;
    V_Sqrt_T = Vol * sqrt(Maturity);
    D = -0.5 * V_Sqrt_T - log(Strike / Forward) / (V_Sqrt_T);
    return Bond * Strike * V_Sqrt_T / Vol * pnl_normal_density(D) ;
}

/**
* gives the second derivative of the price w.r.t to the forward of a call/put
* option in the BS model

```

```

*
* @param Vol a double, the volatility
* @param Bond = exp(-r*Maturity), a double, price of the zero coupon bond
* @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
* time T - maturity.
* @param Strike a double, for value contract
* @param Maturity a double
* @return gamma of a call/put option
*/
double pnl_bs_impli_gamma_forward(double Vol, double Bond, double Forward, double
{
    double V_Sqrt_T;
    double D1;

    PNL_CHECK(Vol < 0., "Volatility required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Maturity < 0., "Maturity required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Forward <= 0., "Maturity required to be > 0", "pnl_bs_impli_put");
    PNL_CHECK(Strike < 0., "Strike required to be >= 0", "pnl_bs_impli_put");

    if (Vol == 0.0 || Maturity == 0.0 || Strike == 0.0) return 0.0;
    V_Sqrt_T = Vol * sqrt(Maturity);
    D1 = 0.5 * V_Sqrt_T - log(Strike / Forward) / (V_Sqrt_T);
    return Bond * pnl_normal_density(D1) / (Forward * V_Sqrt_T) ;
}

/**
* give the gamma times s^2 of a call/put option in BS model so second derivativ
* against spot
* note forward * bond = Spot exp(-divid matu)
*
* @param Vol a double, the volatility
* @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond
* @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
* time T - maturity.
* @param Strike a double, for value contract
* @param Maturity a double, echeance time (T)
* @return s_square_gamma of a call/put option
*/
double pnl_bs_impli_s_square_gamma(double Vol, double Bond, double Forward, double

```

```

{
    double V_Sqrt_T;
    double D1;

    PNL_CHECK(Vol < 0., "Volatility required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Maturity < 0., "Maturity required to be >= 0", "pnl_bs_impli_put");
    PNL_CHECK(Forward <= 0., "Maturity required to be > 0", "pnl_bs_impli_put");
    PNL_CHECK(Strike < 0., "Strike required to be >= 0", "pnl_bs_impli_put");

    if (Vol == 0.0 || Maturity == 0.0 || Strike == 0.0) return 0.0;
    V_Sqrt_T = Vol * sqrt(Maturity);
    D1 = 0.5 * V_Sqrt_T - log(Strike / Forward) / (V_Sqrt_T);
    return Forward * Bond * pnl_normal_density(D1) / V_Sqrt_T;
}

static void pnl_bs_impli_increment_call_put_Type(double x, double *fx, double *dfx, v
{
    *fx = Data->Price - pnl_bs_impli_call_put(Data->is_call, x, Data->Bond, Data->
    *dfx = -1.*pnl_bs_impli_vega(x, Data->Bond, Data->Forward, Data->Strike, Data-
}

static void pnl_bs_impli_increment_call_put(double x, double *fx, double *dfx, v
{
    pnl_bs_impli_increment_call_put_Type(x, fx, dfx, Data);
}

/**
 * compute the implied volatility of an option price
 *
 * @param is_call a int, the option type, 1 for call, 0 for put
 * @param Price a double, option price today
 * @param Bond = exp(-r*Maturity), a double, price of Zero coupon bond
 * @param Forward = Spot*exp((r-divid)*Maturity) a double, price of Spot at
 * time T - maturity.
 * @param Strike a double, for value contract
 * @param Maturity a double, echeance time (T)
 * @return implied of a call/put option
 */
double pnl_bs_impli_implicit_vol(int is_call, double Price, double Bond, double
{
    double impli_vol;

```



```

Pnl_Data_Vol_Impli_BS *data;
PnlFuncDFunc func;

data = malloc(sizeof(Pnl_Data_Vol_Impli_BS));
data->is_call = is_call;
data->Price = Price;
data->Bond = Bond;
data->Forward = Forward;
data->Strike = Strike;
data->Maturity = Maturity;
if (is_call)
{
    if (Price <= Bond * MAX(Forward - Strike, 0.0))
        return FAIL;
    else if (Price >= Bond * Forward)
        return 10.0;
}
else
{
    if (Price <= Bond * MAX(Strike - Forward, 0.0))
        return 0.0;
    else if (Price >= Bond * Strike)
        return 10.0;
}

func.F = pnl_bs_impli_increment_call_put;
func.params = data;
pnl_root_newton_bisection(&func, 0.001, 10.0, 0.0001, 20, &impli_vol);
free(data);
return impli_vol;
}

/**
 * compute implied volatility matrix of a list of options prices
 *
 * @param is_call a int, the option type, 1 for call, 0 for put
 * @param Price a matrix of Prices
 * @param spot the spot price
 * @param rate the instantaneous interest rate
 * @param divid the instantaneous dividend rate

```

```

* @param Strike a Vector, for value contract
* @param Maturity a Vector, echeance time (T)
* @param Vol a Matrix, to store matrix volatility
* @return error code indicator
*/
int pnl_bs_implicit_vol(const PnlMatInt *is_call, const PnlMat *Pri
                        const PnlVect *Strike, const PnlVect *Matur
{
    int i, j;
    int error = 0;
    Pnl_Data_Vol_Impli_BS *data;
    PnlFuncDFunc func;

    data = malloc(sizeof(Pnl_Data_Vol_Impli_BS));
    func.F = pnl_bs_implicit_increment_call_put;
    func.params = data;
    for (j = 0; j < Maturity->size; j++)
    {
        data->Maturity = GET(Maturity, j);
        data->Bond = exp(-rate * data->Maturity);
        data->Forward = spot * exp((rate - divid) * data->Maturity);
        for (i = 0; i < Strike->size; i++)
        {
            data->Price = MGET(Price, i, j);
            data->is_call = pnl_mat_int_get(is_call, i, j);
            data->Strike = GET(Strike, i);

            if (data->is_call)
            {
                if (data->Price <= data->Bond * MAX(data->Forward - data->Strike,
                    MLET(Vol, i, j) = 0.0;
                else if (data->Price >= data->Bond * data->Forward)
                    MLET(Vol, i, j) = 10.0;
            }
        }
    }
    else
    {
        if (data->Price <= data->Bond * MAX(data->Strike - data->Forward,
            MLET(Vol, i, j) = 0.0;
        else if (data->Price >= data->Bond * data->Strike)
            MLET(Vol, i, j) = 10.0;
    }
}

```

```

        error += pnl_root_newton_bisection(&func, 0.001, 10.0, 0.0001, 20, pnl
    }
}
free(data);
return error;
}

static double intrinsic_value(double theta, double x, double (*h)(double))
{
    return h(theta * x) * x * (exp(x * 0.5) - exp(-x * 0.5));
};
static double derivative_b(double x, double sigma)
{
    double y = x / sigma;
    return M_1_SQRT2PI * exp(-0.5 * (y * y + sigma * sigma * 0.25));
}

static double derivative_ln_derivativ_b(double x, double sigma)
{
    double y = x / sigma;
    return y * y / sigma - 0.25 * sigma;
}

static double call(double x)
{
    return MAX(x, 0);
}

void implied_volatility_iterate(double x, double b_m_i, double function_on_b, do
{
    double a = (*nu) / (1 + (*eta));
    double m_sigma_half = -(*sigma) * 0.5;
    double nu_a = function_on_b * derivative_b(x, *sigma);
    (*sigma) += MAX(a, m_sigma_half);
    (*eta) = derivative_ln_derivativ_b(x, *sigma) - nu_a;
    (*nu) = MAX(nu_a, m_sigma_half);
    (*eta) *= (*nu) * 0.5;
    *eta = MAX(*eta, -0.75);
}

```

```

double pnl_bs_impli_implicit_vol_2(int is_call, double Price, double Bond, double
{
    double theta = (is_call) ? 1.0 : -1.0;
    double x = log(Forward / Strike);
    double b = Price / (Bond * sqrt(Forward * Strike));
    double b_m_i = b - intrinsic_value(theta, x, call);
    double ln_b_m_i = log(b_m_i);
    double function_on_b = (2. + ln_b_m_i) / ln_b_m_i * 1. / b_m_i;
    double sigma = 1.0, sigma_anc = 2.0, nu = 0.0, eta = 0.0;
    while (fabs(sigma - sigma_anc) < 1e-13)
    {
        sigma_anc = sigma;
        implied_volatility_iterate(x, b_m_i, function_on_b, &sigma, &nu, &eta);
    }
    return sigma;
}

```