

Help

```

#include "mer1dlocvol_std.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_root.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2014+2) //The "#els
static int CHK_OPT(AP_BGM_MER)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_BGM_MER)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

double Riemann(int n, double T, double *fun)
{
    int i;
    double res, Tn;

    res = 0;
    Tn = T / n;

    for (i = 0; i < n; i++)
    {
        res += fun[i] * Tn;
    }
    return res;
}

/*****
Page 12 Line 2
*****/

int ExpansionTerms(double K, double lambda, double nj, double gammaj, double T,
                    vega_const, double beta_const,
                    double *greek1, double *greek2)

```

```
{
    int i;
    double a1, a2, a3, b1, b2, b3, w1, w2, dT, dT2, FT, t1, Nd1, mean, sd, bound,
    double *vega, *beta, *sigma, *sigma1, *sigma2, *mu, *mu1;
    double intSigma2, d1, vol;
    int which = 1, status;
    int N;
    int dim;

    dim = (int)(T * 20);

    vega = malloc(dim * sizeof(double));
    beta = malloc(dim * sizeof(double));
    sigma = malloc(dim * sizeof(double));
    sigma1 = malloc(dim * sizeof(double));
    sigma2 = malloc(dim * sizeof(double));
    mu = malloc(dim * sizeof(double));
    mu1 = malloc(dim * sizeof(double));

    N = 10;
    mean = 0.0;
    sd = 1.0;
    bound = 0.0;

    total = 0;

    dT = 0.05 * T;
    a1 = 0.0;
    a2 = 0.0;
    a3 = 0.0;
    b1 = 0.0;
    b2 = 0.0;
    b3 = 0.0;
    w1 = 0.0;
    w2 = 0.0;

    for (i = 0; i < dim; i++)
    {
        vega[i] = vega_const - i * 0.0011;
        beta[i] = beta_const - i * 0.0075;
```

```

    /* Page 10 line -3 */
    sigma[i] = vega[i] * exp((beta[i] - 1) * log(S0));
    sigma1[i] = (beta[i] - 1) * sigma[i];
    sigma2[i] = SQR(sigma[i]);
    /* Page 10 line -6 */
    mu[i] = lambda * (1 - exp(nj + SQR(gammaj) * 0.5)) - 0.5 * SQR(sigma[i]);
    mu1[i] = -sigma[i] * sigma1[i];
}

for (i = 0; i < dim; i++)
{
    /* Page 12 line -7 */
    dT2 = SQR((i + 1) * 0.05 * T) - SQR(i * 0.05 * T);
    /* Page 12 line 3 */
    a1 += dT * mu1[i] * w2 + SQR(dT) * 0.5 * mu[i] * mu1[i];
    a2 += dT * (mu1[i] * w1 + sigma[i] * sigma1[i] * w2) + SQR(dT) * 0.5 * (SQ
    a3 += dT * sigma[i] * sigma1[i] * w1 + SQR(dT) * 0.5 * CUB(sigma[i]) * sig
    b1 += lambda * nj * dT2 * 0.5 * sigma1[i];
    b2 += lambda * dT2 * 0.5 * (gammaj * mu1[i] + nj * sigma[i] * sigma1[i]);
    b3 += lambda * gammaj * dT2 * 0.5 * sigma[i] * sigma1[i];
    w1 += dT * SQR(sigma[i]);
    w2 += dT * mu[i];
}

/* Page 9 line 13*/
FT = S0 * exp((r - q) * T + lambda * (1 - exp(nj + 0.5 * SQR(gammaj))) * T);

intSigma2 = Riemann(20, T, sigma2);

for (i = 0; i < N; i++)
{
    t1 = (pnl_pow_i((lambda * T), i) / pnl_fact(i)) * exp(-lambda * T - r * T);
    vol = sqrt(intSigma2 + i * SQR(gammaj) / T);
    d1 = 1 / (vol * sqrt(T)) * (log(FT * exp(i * (nj + 0.5 * SQR(gammaj)))) / K
    pnl_cdf_nor(&which, &Nd1, &q, &d1, &mean, &sd, &status, &bound);
    total += Nd1 * t1;
}

*greek1 = (a1 + a2 + a3) * 1 * total;

total = 0;

```

```

intSigma2 += SQR(gammaj);
for (i = 0; i < N; i++)
{
    t1 = (pnl_pow_i((lambda * T), i) / pnl_fact(i)) * exp(-lambda * T - r * T);
    vol = sqrt(intSigma2 + i * SQR(gammaj) / T);
    d1 = 1 / (vol * sqrt(T)) * (log(FT * exp(i * (nj + 0.5 * SQR(gammaj)))) / K);
    pnl_cdf_nor(&which, &Nd1, &q, &d1, &mean, &sd, &status, &bound);
    total += Nd1 * t1;
}

*greek2 = (b1 + b2 + b3) * 1 * total;

free(vega);
free(beta);
free(sigma);
free(sigma1);
free(sigma2);
free(mu);
free(mu1);

return 0;
}

/*****
Page 9 Line 13
*****/

int Merton(double K, double T, double lambda, double r, double q, double nj, double
    vega_const, double beta_const, double *putPrice)
{
    int N;
    double t1, t2, FT, call, intSigma2;
    int i, dim;
    double *vega, *beta, *sigma, *sigma2;

    dim = (int)(T * 20);

    vega = malloc(dim * sizeof(double));
    beta = malloc(dim * sizeof(double));

```

```

sigma = malloc(dim * sizeof(double));
sigma2 = malloc(dim * sizeof(double));

N = 10;
call = 0;
FT = S0 * exp((r - q) * T + lambda * (1 - exp(nj + 0.5 * SQR(gammaj))) * T);

for (i = 0; i < dim; i++)
{
    vega[i] = vega_const - i * 0.0011;
    beta[i] = beta_const - i * 0.0075;
    sigma[i] = vega[i] * exp((beta[i] - 1) * log(S0));
    sigma2[i] = SQR(sigma[i]);
}

intSigma2 = Riemann(20, T, sigma2);

for (i = 0; i < N; i++)
{
    t1 = (pnl_pow_i((lambda * T), i) / pnl_fact(i)) * exp(-lambda * T - r * T);
    t2 = pnl_bs_call(FT * exp(i * (nj + 0.5 * SQR(gammaj))), K, T, 0, 0, sqrt(
    call += t1 * t2;
}

*putPrice = call;

free(vega);
free(beta);
free(sigma);
free(sigma2);

return 0;
}

static int AP_BGM_Merton(double S0, double T, NumFunc_1 *p, double r, double q,
{
    double Price, term1, term2, K;

    K = p->Par[0].Val.V_PDDOUBLE;

```

```

Merton(K, T, lambda, r, q, nj, gammaj, S0, vega_const, beta_const, &Price);
ExpansionTerms(K, lambda, nj, gammaj, T, S0, r, q, vega_const, beta_const, &te
/* vol=pnl_bs_implicit_vol (1,Price+term1+term2,S0,K,T,r,q,&error); */

*ptprice = Price + term1 + term2;
return OK;
}

int CALC(AP_BGM_MER)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

    return AP_BGM_Merton(ptMod->S0.Val.V_PDOUBLE, ptOpt->Maturity.Val.V_DATE - ptM
}

static int CHK_OPT(AP_BGM_MER)(void *Opt, void *Mod)
{
    return strcmp(((Option *)Opt)->Name, "CallEuro");
}

#endif //PremiaCurrentVersion

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

    return OK;
}

PricingMethod MET(AP_BGM_MER) =
{
    "AP_BGM_MER",
    {" ", PREMIA_NULLTYPE, {0}, FORBID}},

```

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