

Help

```

#include "scott1d_std.h"
#include "enums.h"
#include "error_msg.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_root.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_random.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2012+2) //The "#els
static int CHK_OPT(MC_MultiLevel_Scott)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_MultiLevel_Scott)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static int equal0(double *t)
{
    return *t == 0.;
}

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

static int BS(double r, double rho, double S0, double K, double sig0, double T,
              double nu, PnlVect *vol, PnlVect *z0, PnlVect *y)
{
    PnlVectInt *ind;
    PnlVect *d1, *d2, *z0c, *z0_clone;
    int which, i, A;
    double p;
    double q;
    double x;

```

```
double mean;
double sd;
int status;
double bound;

which = 1;
mean = 0.0;
sd = 1.0;

ind = pnl_vect_int_create(vol->size);

d1 = pnl_vect_create(vol->size);
d2 = pnl_vect_create(vol->size);
z0c = pnl_vect_create(vol->size);
z0_clone = pnl_vect_create(vol->size);

pnl_vect_clone(d1, vol);
pnl_vect_clone(d2, vol);
pnl_vect_clone(z0c, z0);

pnl_vect_find(ind, "v", equal0, vol);

if (ind->size == 0)
{
    //d1=(log(z0/K)+(r+vol.*vol/2)*T)./(vol*sqrt(T));
    pnl_vect_mult_vect_term(d1, d1);
    pnl_vect_mult_double(d1, 0.5 * T);
    pnl_vect_plus_double(d1, r * T);
    pnl_vect_div_double(z0c, K);
    pnl_vect_map_inplace(z0c, log);
    pnl_vect_plus_vect(d1, z0c);
    pnl_vect_div_double(d1, sqrt(T));
    pnl_vect_div_vect_term(d1, vol);
    //d2=d1-vol*sqrt(T);
    pnl_vect_mult_double(d2, -sqrt(T));
    pnl_vect_plus_vect(d2, d1);

    for (i = 0; i < vol->size; i++)
    {
```

```

        x = GET(d1, i);
        pnl_cdf_nor(&which, &p, &q, &x, &mean, &sd, &status, &bound);
        LET(y, i) = p;

    }

    pnl_vect_mult_vect_term(y, z0);

    for (i = 0; i < vol->size; i++)
    {
        x = GET(d2, i);
        pnl_cdf_nor(&which, &p, &q, &x, &mean, &sd, &status, &bound);
        LET(y, i) = GET(y, i) - K * exp(-r * T) * p;

    }
    //y=z0.*cdfnor("PQ",d1,zeros(vol),ones(vol))-K*exp(-r*T)*cdfnor("PQ",d2,ze
}

else
{

    for (i = 0; i < ind->size; i++)
    {
        A = pnl_vect_int_get(ind, i);
        LET(vol, A) = 1.;
    }

    //d1=(log(z0/K)+(r+vol.*vol/2)*T)./(vol*sqrt(T));
    pnl_vect_mult_vect_term(d1, d1);
    pnl_vect_mult_double(d1, 0.5 * T);
    pnl_vect_plus_double(d1, r * T);
    pnl_vect_div_double(z0c, K);
    pnl_vect_map_inplace(z0c, log);
    pnl_vect_plus_vect(d1, z0c);
    pnl_vect_div_double(d1, sqrt(T));
    pnl_vect_div_vect_term(d1, vol);
    //d2=d1-vol*sqrt(T);
    pnl_vect_mult_double(d2, -sqrt(T));
    pnl_vect_plus_vect(d2, d1);

    for (i = 0; i < vol->size; i++)

```

```

    {
        x = GET(d1, i);
        pnl_cdf_nor(&which, &p, &q, &x, &mean, &sd, &status, &bound);
        LET(y, i) = p;
    }

    pnl_vect_mult_vect_term(y, z0);

    for (i = 0; i < vol->size; i++)
    {
        x = GET(d2, i);
        pnl_cdf_nor(&which, &p, &q, &x, &mean, &sd, &status, &bound);
        LET(y, i) = GET(y, i) - K * exp(-r * T) * p;
    }

    for (i = 0; i < ind->size; i++)
    {
        A = pnl_vect_int_get(ind, i);
        LET(y, A) = MAX(0., GET(z0, A) - K * exp(-r * T));
    }

}

pnl_vect_int_free(&ind);
pnl_vect_free(&d1);
pnl_vect_free(&d2);
pnl_vect_free(&z0c);
pnl_vect_free(&z0_clone);

return OK;
}

//Exponential
static int f(double sig0, PnlVect *x, PnlVect *y)
{
    //y=sig0*exp(x);
    pnl_vect_clone(y, x);
    pnl_vect_map_inplace(y, exp);
}

```

```

    pnl_vect_mult_double(y, sig0);

    return OK;
}

static int f2(double sig0, PnlVect *x, PnlVect *y)

{
    //y=sig0^2*exp(2*x);
    pnl_vect_clone(y, x);
    pnl_vect_mult_double(y, 2.);
    pnl_vect_map_inplace(y, exp);
    pnl_vect_mult_double(y, SQR(sig0));

    return OK;
}

//Monte Carlo for a fixed level L
static int mlmc_l(int M, int l, int N, double r, double divid, double rho, doubl

{
    PnlVect *Xf, *Xc, *Mf, *Mc, *yf, *yf0, *clone_yf0, *yc;
    PnlVect *yc0, *int1f, *int2f, *int1c, *int2c, *g1, *g2, *g3, *clone_g1;
    PnlVect *bf, *bf0, *bc, *wf, *wf0, *wc, *f2_yf0_, *f_yf0_;
    PnlVect *volc, *clone_volc, *z0f, *Pf, *volc, *clone_volc;
    PnlVect *z0c, *Pc, *dP, *dPc;
    PnlVect *clone_f2_yf0_, *clone_f_yf0_, *ccclone_f_yf0_;
    PnlVect *f2_yc0_, *f_yc0_, *clone_f2_yc0_;
    PnlVect *clone_f_yc0_, *ccclone_f_yc0_;

    double nf, nc, hf, hc, X0, a, dd, ff;
    int N1, N2, n, m;

    nf = pow(M, l);
    nc = nf / M;
    hf = T / nf;
    hc = T / nc;

    for (N1 = 1; N1 <= N; N1 += 10000)

```

```

{

    N2 = (int) MIN(10000, N - N1 + 1);

    X0 = log(S0);
    Xf = pnl_vect_create_from_double(N2, X0);
    Xc = pnl_vect_create_from_double(N2, 0.);
    Mf = pnl_vect_create_from_double(N2, 0.);
    Mc = pnl_vect_create_from_double(N2, 0.);

    pnl_vect_clone(Xc, Xf);
    pnl_vect_clone(Mf, Xf);
    pnl_vect_clone(Mc, Xc);

    pnl_vect_map_inplace(Mf, exp);
    pnl_vect_map_inplace(Mc, exp);
    yf = pnl_vect_create_from_double(N2, 0.);
    yf0 = pnl_vect_create_from_double(N2, 0.);
    yc = pnl_vect_create_from_double(N2, 0.);
    yc0 = pnl_vect_create_from_double(N2, 0.);
    clone_yf0 = pnl_vect_create_from_double(N2, 0.);

    pnl_vect_clone(yc, yf);
    bf = pnl_vect_create_from_double(N2, 0.);
    bf0 = pnl_vect_create_from_double(N2, 0.);
    bc = pnl_vect_create_from_double(N2, 0.);
    wf = pnl_vect_create_from_double(N2, 0.);
    wf0 = pnl_vect_create_from_double(N2, 0.);
    wc = pnl_vect_create_from_double(N2, 0.);

    int1f = pnl_vect_create_from_double(N2, 0.);
    int2f = pnl_vect_create_from_double(N2, 0.);
    int1c = pnl_vect_create_from_double(N2, 0.);
    int2c = pnl_vect_create_from_double(N2, 0.);

    volf = pnl_vect_create_from_double(N2, 0.);
    clone_volf = pnl_vect_create_from_double(N2, 0.);

    volc = pnl_vect_create_from_double(N2, 0.);
    clone_volc = pnl_vect_create_from_double(N2, 0.);

```

```

z0f = pnl_vect_create_from_double(N2, 0.);
Pf = pnl_vect_create_from_double(N2, 0.);

z0c = pnl_vect_create_from_double(N2, 0.);
Pc = pnl_vect_create_from_double(N2, 0.);

dP = pnl_vect_create_from_double(N2, 0.);

a = nu * sqrt((1 - exp(-2 * kappa * hf)) / (2 * kappa));
dd = nu * (1 - exp(-kappa * hf)) / (kappa * a);
ff = sqrt(hf - SQR(dd));

g1 = pnl_vect_create_from_double(N2, 0);
clone_g1 = pnl_vect_create_from_double(N2, 0);
g2 = pnl_vect_create_from_double(N2, 0);
g3 = pnl_vect_create_from_double(N2, 0);

f2_yf0_ = pnl_vect_create_from_double(N2, 0);
f_yf0_ = pnl_vect_create_from_double(N2, 0);

clone_f2_yf0_ = pnl_vect_create_from_double(N2, 0);
clone_f_yf0_ = pnl_vect_create_from_double(N2, 0);
cclone_f_yf0_ = pnl_vect_create_from_double(N2, 0);

f2_yc0_ = pnl_vect_create_from_double(N2, 0);
f_yc0_ = pnl_vect_create_from_double(N2, 0);

clone_f2_yc0_ = pnl_vect_create_from_double(N2, 0);
clone_f_yc0_ = pnl_vect_create_from_double(N2, 0);
cclone_f_yc0_ = pnl_vect_create_from_double(N2, 0);

if (l == 0)
{
    pnl_vect_rand_normal(g1, N2, gen);
    pnl_vect_rand_normal(g2, N2, gen);
    pnl_vect_clone(yf0, yf);
    pnl_vect_clone(wf, g1);
    pnl_vect_clone(bf, g2);

    pnl_vect_mult_double(wf, sqrt(hf));

```

```

    pnl_vect_mult_double(bf, sqrt(hf));
    //Xf=Xf+(r-f2_yf0_/2)*hf+rho*f_yf0_.*wf+sqrt(1-rho^2)*f_yf0_.*bf;
    f2(sig0, yf0, f2_yf0_);
    pnl_vect_clone(clone_f2_yf0_, f2_yf0_);
    pnl_vect_mult_double(clone_f2_yf0_, -0.5);
    pnl_vect_plus_double(clone_f2_yf0_, r);
    pnl_vect_mult_double(clone_f2_yf0_, hf);
    f(sig0, yf0, f_yf0_);

    pnl_vect_clone(clone_f_yf0_, f_yf0_);
    pnl_vect_mult_vect_term(clone_f_yf0_, wf);
    pnl_vect_mult_double(clone_f_yf0_, rho);

    pnl_vect_clone(cclone_f_yf0_, f_yf0_);
    pnl_vect_mult_vect_term(cclone_f_yf0_, bf);
    pnl_vect_mult_double(cclone_f_yf0_, sqrt(1 - SQR(rho)));
    pnl_vect_plus_vect(cclone_f_yf0_, clone_f_yf0_);
    pnl_vect_plus_vect(cclone_f_yf0_, clone_f2_yf0_);
    pnl_vect_plus_vect(Xf, cclone_f_yf0_);

    //int1f=int1f+(-f2(yf0)/2)*hf+rho*f(yf0).*wf;
    pnl_vect_clone(clone_f2_yf0_, f2_yf0_);
    pnl_vect_mult_double(clone_f2_yf0_, -0.5);
    pnl_vect_mult_double(clone_f2_yf0_, hf);

    pnl_vect_plus_vect(clone_f2_yf0_, clone_f_yf0_);
    pnl_vect_plus_vect(int1f, clone_f2_yf0_);

    //int2f=int2f+f2(yf0);
    pnl_vect_plus_vect(int2f, f2_yf0_);

}

else

{
    for (n = 1; n <= (int)(nc); n++)
    {
        for (m = 1; m <= M; m++)

            {

```

```

pnl_vect_clone(yf0, yf);
pnl_vect_clone(bf0, bf);
pnl_vect_clone(wf0, wf);
pnl_vect_rand_normal(g1, N2, gen);
pnl_vect_rand_normal(g2, N2, gen);
pnl_vect_rand_normal(g3, N2, gen);

//yf=exp(-kappa*hf)*yf0+theta*(1-exp(-kappa*hf))+a*g1;
pnl_vect_clone(clone_g1, g1);
pnl_vect_mult_double(clone_g1, a);
pnl_vect_plus_double(clone_g1, theta * (1 - exp(-kappa * hf)));
pnl_vect_clone(clone_yf0, yf0);
pnl_vect_mult_double(clone_yf0, exp(-kappa * hf));
pnl_vect_plus_vect(clone_g1, clone_yf0);
pnl_vect_clone(yf, clone_g1);
//wf=dd*g1+ff*g2
pnl_vect_mult_double(g1, dd);
pnl_vect_mult_double(g2, ff);
pnl_vect_plus_vect(g1, g2);
pnl_vect_clone(wf, g1);
//bf=sqrt(hf)*g3;
pnl_vect_mult_double(g3, sqrt(hf));
pnl_vect_clone(bf, g3);
//Xf=Xf+(r-f2(yf0)/2)*hf+rho*f(yf0).*wf+sqrt(1-rho^2)*f(yf0).*
f2(sig0, yf0, f2_yf0_);
pnl_vect_clone(clone_f2_yf0_, f2_yf0_);
pnl_vect_mult_double(clone_f2_yf0_, -0.5);
pnl_vect_plus_double(clone_f2_yf0_, r);
pnl_vect_mult_double(clone_f2_yf0_, hf);
f(sig0, yf0, f_yf0_);

pnl_vect_clone(clone_f_yf0_, f_yf0_);
pnl_vect_mult_vect_term(clone_f_yf0_, wf);
pnl_vect_mult_double(clone_f_yf0_, rho);

pnl_vect_clone(cclone_f_yf0_, f_yf0_);
pnl_vect_mult_vect_term(cclone_f_yf0_, bf);
pnl_vect_mult_double(cclone_f_yf0_, sqrt(1 - SQR(rho)));
pnl_vect_plus_vect(cclone_f_yf0_, clone_f_yf0_);
pnl_vect_plus_vect(cclone_f_yf0_, clone_f2_yf0_);

```

```

    pnl_vect_plus_vect(Xf, cclone_f_yf0_);

    //int1f=int1f+(-f2_yf0_/2)*hf+rho*f(yf0).*wf;
    pnl_vect_clone(clone_f2_yf0_, f2_yf0_);
    pnl_vect_mult_double(clone_f2_yf0_, -0.5);
    pnl_vect_mult_double(clone_f2_yf0_, hf);

    pnl_vect_plus_vect(clone_f2_yf0_, clone_f_yf0_);
    pnl_vect_plus_vect(int1f, clone_f2_yf0_);

    //int2f=int2f+f2_yf0_;
    pnl_vect_plus_vect(int2f, f2_yf0_);

}

pnl_vect_clone(yc0, yc);
pnl_vect_clone(yc, yf);
//bc=bf0+bf;
pnl_vect_plus_vect(bf0, bf);
pnl_vect_clone(bc, bf0);
//wc=wf0+wf;
pnl_vect_plus_vect(wf0, wf);
pnl_vect_clone(wc, wf0);
//Xc=Xc+(r-f2(yc0)/2)*hc+rho*f(yc0).*wc+sqrt(1-rho^2)*f(yc0).*bc;
f2(sig0, yc0, f2_yc0_);
pnl_vect_clone(clone_f2_yc0_, f2_yc0_);
pnl_vect_mult_double(clone_f2_yc0_, -0.5);
pnl_vect_plus_double(clone_f2_yc0_, r);
pnl_vect_mult_double(clone_f2_yc0_, hc);
f(sig0, yc0, f_yc0_);

pnl_vect_clone(clone_f_yc0_, f_yc0_);
pnl_vect_mult_vect_term(clone_f_yc0_, wc);
pnl_vect_mult_double(clone_f_yc0_, rho);

pnl_vect_clone(cclone_f_yc0_, f_yc0_);
pnl_vect_mult_vect_term(cclone_f_yc0_, bc);
pnl_vect_mult_double(cclone_f_yc0_, sqrt(1 - SQR(rho)));

```

```

    pnl_vect_plus_vect(ccclone_f_yc0_, clone_f_yc0_);
    pnl_vect_plus_vect(ccclone_f_yc0_, clone_f2_yc0_);
    pnl_vect_plus_vect(Xc, cclone_f_yc0_);

    //int1c=int1c+(-f2(yc0)/2)*hc+rho*f(yc0).*wc;

    pnl_vect_clone(clone_f2_yc0_, f2_yc0_);
    pnl_vect_mult_double(clone_f2_yc0_, -0.5);
    pnl_vect_mult_double(clone_f2_yc0_, hc);

    pnl_vect_plus_vect(clone_f2_yc0_, clone_f_yc0_);
    pnl_vect_plus_vect(int1c, clone_f2_yc0_);

    //int2c=int2c+f2(yc0);
    pnl_vect_plus_vect(int2c, f2_yc0_);
}

}

//volf=sqrt(1-rho^2)*sqrt(hf*int2f)/sqrt(T);
pnl_vect_mult_double(int2f, hf);
pnl_vect_map_inplace(int2f, sqrt);
pnl_vect_mult_double(int2f, sqrt(1 - SQR(rho)) / sqrt(T));
pnl_vect_clone(volf, int2f);

//z0f=S0*exp(int1f-q+(volf.*volf/2)*T);
pnl_vect_clone(clone_volf, volf);
pnl_vect_mult_vect_term(clone_volf, clone_volf);
pnl_vect_mult_double(clone_volf, T / 2);
pnl_vect_plus_vect(clone_volf, int1f);
pnl_vect_plus_double(clone_volf, -divid);
pnl_vect_map_inplace(clone_volf, exp);
pnl_vect_mult_double(clone_volf, S0);
pnl_vect_clone(z0f, clone_volf);

//Pf=BS(z0f,volf);
BS(r, rho, S0, K, sig0, T, theta, kappa, nu, volf, z0f, Pf);

```

```

//volc=sqrt(1-rho^2)*sqrt(hc*int2c)/sqrt(T);
pnl_vect_mult_double(int2c, hc);
pnl_vect_map_inplace(int2c, sqrt);
pnl_vect_mult_double(int2c, sqrt(1 - SQR(rho)) / sqrt(T));
pnl_vect_clone(volc, int2c);
//z0c=S0*exp(int1c+(volc.*volc/2)*T);
pnl_vect_clone(clone_volc, volc);
pnl_vect_mult_vect_term(clone_volc, clone_volc);
pnl_vect_mult_double(clone_volc, T / 2);
pnl_vect_plus_vect(clone_volc, int1c);
pnl_vect_plus_double(clone_volc, -divid);
pnl_vect_clone(z0c, clone_volc);

pnl_vect_map_inplace(z0c, exp);
pnl_vect_mult_double(z0c, S0);
//Pc=BS(z0c,volc);
BS(r, rho, S0, K, sig0, T, theta, kappa, nu, volc, z0c, Pc);

//dP=Pf-Pc;
pnl_vect_mult_double(Pc, -1);
pnl_vect_plus_vect(Pc, Pf);
pnl_vect_clone(dP, Pc);

//Pc2 = Pc;
//dPc = exp(-r*T)*(Pc2-Pc);
dPc = pnl_vect_create_from_double(N2, 0);

if (l == 0)
{
    pnl_vect_clone(dP, Pf);
}

//sums(1) = sums(1) + sum(dP);
LET(sums, 0) = GET(sums, 0) + pnl_vect_sum(dP);
//sums(2) = sums(2) + sum(dP.^2);
pnl_vect_mult_vect_term(dP, dP);
LET(sums, 1) = GET(sums, 1) + pnl_vect_sum(dP);

```

```

//sums(3) = sums(3) + sum(Pf);
LET(sums, 2) = GET(sums, 2) + pnl_vect_sum(Pf);
//sums(4) = sums(4) + sum(Pf.^2);
pnl_vect_mult_vect_term(Pf, Pf);
LET(sums, 3) = GET(sums, 3) + pnl_vect_sum(Pf);
//sums(5) = sums(5) + sum(dPc);
LET(sums, 4) = GET(sums, 4) + pnl_vect_sum(dPc);
//sums(6) = sums(6) + sum(dPc.^2);
LET(sums, 5) = GET(sums, 5) + pnl_vect_sum(dPc);

```

```

pnl_vect_free(&Xf);
pnl_vect_free(&Xc);
pnl_vect_free(&Mf);
pnl_vect_free(&Mc);
pnl_vect_free(&yf);
pnl_vect_free(&yf0);
pnl_vect_free(&clone_yf0);
pnl_vect_free(&yc);
pnl_vect_free(&yc0);
pnl_vect_free(&int1f);
pnl_vect_free(&int2f);
pnl_vect_free(&int1c);
pnl_vect_free(&int2c);
pnl_vect_free(&g1);
pnl_vect_free(&clone_g1);
pnl_vect_free(&g2);
pnl_vect_free(&g3);
pnl_vect_free(&bf);
pnl_vect_free(&bf0);
pnl_vect_free(&bc);
pnl_vect_free(&wf);
pnl_vect_free(&wf0);
pnl_vect_free(&wc);
pnl_vect_free(&volf);
pnl_vect_free(&clone_volf);
pnl_vect_free(&z0f);
pnl_vect_free(&Pf);
pnl_vect_free(&volc);
pnl_vect_free(&clone_volc);

```

```

        pnl_vect_free(&z0c);
        pnl_vect_free(&Pc);
        pnl_vect_free(&dP);
        pnl_vect_free(&dPc);
        pnl_vect_free(&f2_yf0_);
        pnl_vect_free(&f_yf0_);
        pnl_vect_free(&clone_f2_yf0_);
        pnl_vect_free(&clone_f_yf0_);
        pnl_vect_free(&cclone_f_yf0_);
        pnl_vect_free(&f2_yc0_);
        pnl_vect_free(&f_yc0_);
        pnl_vect_free(&clone_f2_yc0_);
        pnl_vect_free(&clone_f_yc0_);
        pnl_vect_free(&cclone_f_yc0_);

    }

    return OK;
}

int MCMultiLevelScott(double S0, NumFunc_1 *p, double T, double r, double divid
{
    double K, sig0;
    PnlVect *sums, *Vl, *clone_Vl, *row_suml_0, *row_suml_1, *row_suml_2, *ML, *NL;
    PnlMat *suml, *trans_suml;
    double L, SOMME, dNl, max_abs, suml_0_i, suml_1_i, P;
    int i, l;
    int converged;
    int M, flag_call;

    M = 2;

    //Parametrization given in Jourdain Sbai paper. SEE Documentation.
    sig0 = exp(v0);
    theta = theta - v0;

    K = p->Par[0].Val.V_PDOUBLE;
    if ((p->Compute) == &Call)

```

```
    flag_call = 1;
else
    flag_call = 0;;

pnl_rand_init(gen, 1, N);

L = -1;

converged = 0;
max_abs = 0;
suml_0_i = 0;
suml_1_i = 0;

suml = pnl_mat_create(0, 0);
trans_suml = pnl_mat_create(0, 0);

// initial variance estimate

Vl =      pnl_vect_create(0);
clone_Vl = pnl_vect_create(0);
Nl =      pnl_vect_create(0);
row_suml_0 = pnl_vect_create(0);
row_suml_1 = pnl_vect_create(0);
row_suml_2 = pnl_vect_create(0);
ML =      pnl_vect_create(0);
sums =    pnl_vect_create_from_double(6, 0.);
con =     pnl_vect_create_from_double(2, 0.);

while (!converged)
{
    pnl_vect_set_zero(Vl);
    pnl_vect_set_zero(clone_Vl);
    pnl_vect_set_zero(Nl);
    pnl_vect_set_zero(row_suml_0);
    pnl_vect_set_zero(row_suml_1);
    pnl_vect_set_zero(row_suml_2);
    pnl_vect_set_zero(ML);
    pnl_vect_set_zero(sums);
    pnl_vect_set_zero(con);
```

```

pnl_mat_tr(trans_suml, suml);

L = L + 1;

pnl_mat_resize(trans_suml, (int)(L + 1), 3);
pnl_mat_resize(suml, 3, (int)(L + 1));
pnl_mat_tr(suml, trans_suml);

pnl_vect_resize(Vl, (int)(L + 1));
pnl_vect_resize(clone_Vl, (int)(L + 1));
pnl_vect_resize(Nl, (int)(L + 1));
pnl_vect_resize(row_suml_0, (int)(L + 1));
pnl_vect_resize(row_suml_1, (int)(L + 1));
pnl_vect_resize(row_suml_2, (int)(L + 1));
pnl_vect_resize(ML, (int)(L + 1));

mlmc_l(M, (int)(L), N, r, divid, rho, S0, K, sig0, T, theta, kappa,

pnl_mat_set(suml, 0, L, (double)(N));
pnl_mat_set(suml, 1, L, GET(sums, 0));
pnl_mat_set(suml, 2, L, GET(sums, 1));

// optimal sample sizes
//Vl = max(0, suml(3,:)./suml(1,:) - (suml(2,:)./suml(1,:)).^2);

pnl_mat_get_row(row_suml_0, suml, 0);
pnl_mat_get_row(row_suml_1, suml, 1);
pnl_mat_get_row(row_suml_2, suml, 2);

pnl_vect_div_vect_term(row_suml_1, row_suml_0);
pnl_vect_mult_vect_term(row_suml_1, row_suml_1);
pnl_vect_mult_double(row_suml_1, -1);
pnl_vect_div_vect_term(row_suml_2, row_suml_0);

pnl_vect_plus_vect(row_suml_2, row_suml_1);

pnl_vect_map_inplace(row_suml_2, max_vector);
pnl_vect_clone(Vl, row_suml_2);

```

```

//Nl = ceil(2*sqrt(Vl./(M.^(0:L)))*sum(sqrt(Vl.*(M.^(0:L))))/eps^2);

for (i = 0; i <= (int)(L); i++)
{

    LET(ML, i) = pow(M, i);

}

pnl_vect_clone(clone_Vl, Vl);
pnl_vect_div_vect_term(clone_Vl, ML);
pnl_vect_map_inplace(clone_Vl, sqrt);

SOMME = 0;
pnl_vect_mult_vect_term(Vl, ML);
pnl_vect_map_inplace(Vl, sqrt);
SOMME = pnl_vect_sum(Vl);
SOMME = SOMME / SQR(eps);

pnl_vect_mult_double(clone_Vl, 2 * SOMME);
pnl_vect_map_inplace(clone_Vl, ceil);

// Nl = max(Nl,N);
pnl_vect_clone(Nl, clone_Vl);
pnl_vect_plus_double(Nl, -N);
pnl_vect_map_inplace(Nl, max_vector);
pnl_vect_plus_double(Nl, N);

// update sample sums

for (l = 0; l <= (int)(L); l++)
{
    //dNl = Nl(l+1)-suml(1,l+1);
    dNl = GET(Nl, l) - MGET(suml, 0, l);

    if (dNl > 0)
    {

        pnl_vect_set_zero(sums);
    }
}

```

```

        mlmc_l(M, l, (int)(dNl), r, divid, rho, S0, K, sig0, T, theta

        MLET(suml, 0, l) = MGET(suml, 0, l) + dNl;
        MLET(suml, 1, l) = MGET(suml, 1, l) + GET(sums, 0);
        MLET(suml, 2, l) = MGET(suml, 2, l) + GET(sums, 1);
    }

}

// test for convergence
rang = pnl_vect_create_from_list(2, -1., 0.);

if ((L > 1) & (pow(M, L) >= 16))
{
    //con = M.^rang.*suml(2,L+1+rang)./suml(1,L+1+rang);
    for (i = 0; i <= 1; i++)
    {
        LET(con, i) = pow(M, GET(rang, i)) * MGET(suml, 1, (int)(L + GET(rang, i)));
    }

    pnl_vect_map_inplace(con, fabs);
    max_abs = pnl_vect_max(con);
    converged = ((max_abs < (M - 1) * eps / sqrt(2)) | (pow(M, L) > 1024.))
}
pnl_vect_free(&rang);

}

pnl_vect_free(&Vl);
pnl_vect_free(&clone_Vl);
pnl_vect_free(&row_suml_0);

```

```

pnl_vect_free(&row_suml_1);
pnl_vect_free(&row_suml_2);
pnl_vect_free(&ML);
pnl_vect_free(&Nl);
pnl_vect_free(&con);
pnl_vect_free(&sums);

// evaluate multi-timestep estimator
// P = sum( suml(2,1:L+1)./suml(1,1:L+1) );

P = 0;

for (i = 0; i <= (int)(L); i++)
{
    suml_1_i = MGET(suml, 1, i);
    suml_0_i = MGET(suml, 0, i);
    P = P + suml_1_i / suml_0_i;
}

pnl_mat_free(&suml);
pnl_mat_free(&trans_suml);

/* Price Call */
*ptprice = P;

//Put Case
if (flag_call == 0)
{
    *ptprice = *ptprice - S0 * exp(-divid * T) + K * exp(-r * T);
}

return OK;
}

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

```

```

if (ptMod->Sigma.Val.V_PDOUBLE == 0.0)
{
    Fprintf(TOSCREEN, "BLACK-SHOLES MODEL\ n\ n\ n");
    return WRONG;
}
else
{
    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

    return MCMultiLevelScott(ptMod->S0.Val.V_PDOUBLE,
                             ptOpt->PayOff.Val.V_NUMFUNC_1,
                             ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                             r,
                             divid, ptMod->Sigma0.Val.V_PDOUBLE
                             , ptMod->MeanReversion.Val.V_PDOUBLE,
                             ptMod->LongRunVariance.Val.V_PDOUBLE,
                             ptMod->Sigma.Val.V_PDOUBLE,
                             ptMod->Rho.Val.V_PDOUBLE,
                             Met->Par[0].Val.V_LONG,
                             Met->Par[1].Val.V_ENUM.value,
                             Met->Par[2].Val.V_PDOUBLE,
                             &(Met->Res[0].Val.V_DOUBLE)
                             );
}
}

static int CHK_OPT(MC_MultiLevel_Scott)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt)

        return OK;
    return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {

```

```

    Met->init = 1;

    Met->Par[0].Val.V_LONG = 1000;
    Met->Par[1].Val.V_ENUM.value = 0;
    Met->Par[1].Val.V_ENUM.members = &PremiaEnumMCRNGs;
    Met->Par[2].Val.V_PDOUBLE = 0.05;
}

return OK;
}

PricingMethod MET(MC_MultiLevel_Scott) =
{
    "MC_MultiLevel",
    { {"N iterations", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {"Accuracy", DOUBLE, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_MultiLevel_Scott),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_MultiLevel_Scott),
    CHK_mc,
    MET(Init)
};

```