

[Help](#)

```
#include "
href../../../../mod/bsnd/bsnd_stdnd/bsnd_stdnd_h_src.pdfbsnd_stdnd.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_vector.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2013+2) //The "#els
static int CHK_OPT(AP_MultiSpread)(void *Opt, void *Mod)
{
    return NONACTIVE;
}

int CALC(AP_MultiSpread)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

// Trace operator
static double trace(PnlMat *m)
{
    int i;
    double sum = 0.0;
    for (i = 0; i < m->n; i++)
    {
        sum += pnl_mat_get(m, i, i);
    }
    return sum;
}

// Matrix square norm tr(M*M')
static double matnorm(const PnlMat *M)
{
    int i, j;
    double m = 0.;

    for (i = 0 ; i < M->m ; i++)
    {
```

```

        for (j = 0 ; j < M->n ; j++)
        {
            m += MGET(M, i, j) * MGET(M, j, i);
        }
    }
    return m;
}

```

```

// Square root of a matrix: r*r=m
static void matsqrt(const PnlMat *m, PnlMat *r)
{
    PnlMat *inv_r, *m_inv_r, *sub_r2_m;
    int N = 0;
    pnl_mat_clone(r, m);
    inv_r = pnl_mat_copy(m);
    m_inv_r = pnl_mat_copy(m);
    sub_r2_m = pnl_mat_mult_mat(r, r);
    pnl_mat_minus_mat(sub_r2_m, m);

    while ((matnorm(sub_r2_m) > 1e-10) && (N < 1e5))
    {
        pnl_mat_inverse(inv_r, r);
        pnl_mat_mult_mat_inplace(m_inv_r, m, inv_r);
        pnl_mat_plus_mat(r, m_inv_r);
        pnl_mat_div_double(r, 2.0);
        pnl_mat_mult_mat_inplace(sub_r2_m, r, r);
        pnl_mat_minus_mat(sub_r2_m, m);
        N++;
    }
    pnl_mat_free(&inv_r);
    pnl_mat_free(&m_inv_r);
    pnl_mat_free(&sub_r2_m);
}

```

```

static void ap_spread_li(PnlVect *s0, double K, double t, double r, PnlVect *div
                        PnlVect *sigma, PnlVect *ptdelta, double rho, int flag_
                        double *ptprice)
{
    int i, j;
    // Matrices and vectors involved in the computations

```

```

PnlMat *Sigma11, *cholSigma11, *sqrts11, *d2x0, *F, *E, *interMM, *interFF;
PnlVect *dd_V, *d_V, *interM1_V, *interM2_V, *Sigma10_V, *s11is10_V, *dx0_V, *

double sSigmaxy, R, cc, c, c_last, *mu, *nu, *emu, psi, sqrtpsi, j0, j1, j2, I
//Intermediary variables
double vfv, uu, phiusqrtpsi, inter1, inter2, res = 0.0;

// dimension = N+1
int const N = s0->size - 1;
mu = malloc((N + 1) * sizeof(double));
nu = malloc((N + 1) * sizeof(double));
emu = malloc((N + 1) * sizeof(double));

// Create and fill the correlation matrix
Sigma11 = pnl_mat_create(N, N);
sqrts11 = pnl_mat_create(N, N);
dx0_V = pnl_vect_create(N);
Sigma10_V = pnl_vect_create(N);
for (i = 0; i < N; i++)
{
    for (j = 0; j < N; j++)
    {
        if (i == j)
        {
            pnl_mat_set(Sigma11, i , j , 1.0);
        }
        else
        {
            pnl_mat_set(Sigma11, i , j , rho);
        }
    }
    if (i < N)
    {
        pnl_vect_set(Sigma10_V, i , rho);
    }
}

s11is10_V = pnl_vect_create(N);
d2x0 = pnl_mat_create(N, N);

```

```

cholSigma11 = pnl_mat_copy(Sigma11);
pnl_mat_chol(cholSigma11);
pnl_mat_chol_syslin(s11is10_V, cholSigma11, Sigma10_V);

sSigmaxy = sqrt(1.0 - pnl_vect_scalar_prod(s11is10_V, Sigma10_V));

//5
for (i = 0; i <= N; i++)
{
    inter1 = pnl_vect_get(sigma, i);
    nu[i] = inter1 * sqrt(t);
    inter1 = pnl_vect_get(divid, i);
    inter2 = pnl_vect_get(s0, i);
    mu[i] = log(inter2) + (r - inter1) * t - 0.5 * nu[i] * nu[i];
    emu[i] = exp(mu[i]);
}

//prop 3
R = 0.0;
for (i = 1; i <= N; i++)
{
    R += emu[i];
}

for (i = 0; i < N; i++)
{
    pnl_vect_set(dx0_V, i, emu[i + 1]*nu[i + 1] / (nu[0] * (R + K)));
}

for (i = 0; i < N; i++)
{
    for (j = 0; j < i; j++)
    {
        inter1 = pnl_vect_get(dx0_V, i);
        inter2 = pnl_vect_get(dx0_V, j);
        pnl_mat_set(d2x0, i, j, -inter1 * inter2 * nu[0]);
        pnl_mat_set(d2x0, j, i, -inter1 * inter2 * nu[0]);
    }
    inter1 = pnl_vect_get(dx0_V, i);
    pnl_mat_set(d2x0, i, i,

```

```

        -inter1 * inter1 * nu[0] + nu[i + 1]*nu[i + 1]*emu[i + 1] / (n
    }

//13, c in the paper
cc = -(log(R + K) - mu[0]) / (nu[0] * sSigmaxy);

dd_V = pnl_vect_copy(s11is10_V);
pnl_vect_minus_vect(dd_V, dx0_V);
pnl_vect_div_double(dd_V, sSigmaxy);

//15
E = pnl_mat_copy(d2x0);
pnl_mat_mult_double(E, -.5 / sSigmaxy);

//29
matsqrt(Sigma11, sqrts11);
interMM = pnl_mat_mult_mat(E, sqrts11);
F = pnl_mat_mult_mat(sqrts11, interMM);

//27 c_{N+1} in the paper
c_last = cc + trace(F);

//28 d_{N+1} in the paper
d_last_V = pnl_mat_mult_vect(sqrts11, dd_V);

//23
interM1_V = pnl_mat_mult_vect(E, Sigma10_V);
pnl_vect_mult_double(interM1_V, nu[0]);
pnl_vect_plus_vect(interM1_V, dd_V);
c = c_last + nu[0] * (sSigmaxy + pnl_vect_scalar_prod(interM1_V, Sigma10_V));

//24
pnl_mat_mult_vect_inplace(interM1_V, E, Sigma10_V);
pnl_vect_mult_double(interM1_V, 2.*nu[0]);
d_V = pnl_mat_mult_vect(sqrts11, interM1_V);
pnl_vect_plus_vect(d_V, d_last_V);

//30
psi = 1. / (1.0 + pnl_vect_scalar_prod(d_V, d_V));

```

```

sqrtpsi = sqrt(psi);

//35
j0 = pnl_cdfnor(c * sqrtpsi);
pnl_mat_mult_vect_inplace(interM1_V, F, d_V);
vfv = pnl_vect_scalar_prod(d_V, interM1_V);
uu = c * c;
phiusqrtpsi = pnl_normal_density(c * sqrtpsi);

//36
j1 = psi * sqrtpsi * (psi * uu - 1.0) * vfv * phiusqrtpsi;

//37
interM2_V = pnl_vect_copy(interM1_V);
pnl_mat_mult_vect_inplace(interM1_V, F, interM2_V);
j2 = c * psi * sqrtpsi * phiusqrtpsi;
interFF = pnl_mat_mult_mat(F, F);
j2 *= 2.0 * trace(interFF) - 4.0 * (1.0 - trace(F)) * psi * (1.0 - psi) * vfv
      psi * psi * (9.0 + (2.0 - 3.0 * uu) * psi - uu * (4.0 - uu) * psi * psi)
      2.0 * psi * (5.0 + (1.0 - 2.0 * uu) * psi) * pnl_vect_scalar_prod(d_V, i

//17
I = j0 + j1 - 0.5 * j2;

res += exp(-r * t + mu[0] + .5 * nu[0] * nu[0]) * I;

inter1 = exp(-t * pnl_vect_get(divid, 0));
if (flag_app > 0)
{
    pnl_vect_set(ptdelta, 0 , I * inter1);
}
else
{
    pnl_vect_set(ptdelta, 0 , I * inter1 - 1.0);
}
canoni = pnl_vect_create(N);

//For 1 to N terms
for (i = 0; i < N; i++)

```

```

{

    pnl_vect_set_zero(canoni);
    pnl_vect_set(canoni, i, 1.0);

    pnl_mat_mult_vect_inplace(interM1_V, Sigma11, canoni);
    pnl_mat_mult_vect_inplace(interM2_V, E, interM1_V);
    pnl_vect_mult_double(interM2_V, nu[i + 1]);
    pnl_vect_plus_vect(interM2_V, dd_V);
    c = c_last + nu[i + 1] * pnl_vect_scalar_prod(interM1_V, interM2_V);

    //24
    pnl_mat_mult_vect_inplace(interM2_V, E, interM1_V);
    pnl_vect_mult_double(interM2_V, 2.*nu[i + 1]);
    pnl_mat_mult_vect_inplace(d_V, sqrts11, interM2_V);
    pnl_vect_plus_vect(d_V, d_last_V);

    //30
    psi = 1. / (1.0 + pnl_vect_scalar_prod(d_V, d_V));
    sqrtpsi = sqrt(psi);

    //35
    j0 = pnl_cdfnor(c * sqrtpsi);
    pnl_mat_mult_vect_inplace(interM1_V, F, d_V);
    vfv = pnl_vect_scalar_prod(d_V, interM1_V);
    uu = c * c;
    phiusqrtpsi = pnl_normal_density(c * sqrtpsi);

    //36
    j1 = psi * sqrtpsi * (psi * uu - 1.0) * vfv * phiusqrtpsi;

    //37
    pnl_vect_clone(interM2_V, interM1_V);
    pnl_mat_mult_vect_inplace(interM1_V, F, interM2_V);
    j2 = c * psi * sqrtpsi * phiusqrtpsi;
    pnl_mat_mult_mat_inplace(interFF, F, F);
    j2 *= 2.0 * trace(interFF) - 4.0 * (1.0 - trace(F)) * psi * (1.0 - psi) *
        psi * psi * (9.0 + (2.0 - 3.0 * uu) * psi - uu * (4.0 - uu) * psi *
        2.0 * psi * (5.0 + (1.0 - 2.0 * uu) * psi) * pnl_vect_scalar_prod(d_

I = j0 + j1 - 0.5 * j2;

```

```

    res -= exp(-r * t + mu[i + 1] + .5 * nu[i + 1] * nu[i + 1]) * I;

    inter1 = exp(-t * pnl_vect_get(divid, i + 1));
    if (flag_app > 0)
    {
        pnl_vect_set(ptdelta, i + 1 , -I * inter1);
    }
    else
    {
        pnl_vect_set(ptdelta, i + 1 , 1.0 - I * inter1);
    }
}

//the last term of the sum

//27
c = c_last;

//28
pnl_vect_clone(d_V, d_last_V);

//30
psi = 1. / (1.0 + pnl_vect_scalar_prod(d_V, d_V));
sqrtpsi = sqrt(psi);

//35
j0 = pnl_cdfnor(c * sqrtpsi);

pnl_mat_mult_vect_inplace(interM1_V, F, d_V);
vfv = pnl_vect_scalar_prod(d_V, interM1_V);
uu = c * c;
phiusqrtpsi = pnl_normal_density(c * sqrtpsi);

//36
j1 = psi * sqrtpsi * (psi * uu - 1.0) * vfv * phiusqrtpsi;

//37
pnl_vect_clone(interM2_V, interM1_V);
pnl_mat_mult_vect_inplace(interM1_V, F, interM2_V);
j2 = c * psi * sqrtpsi * phiusqrtpsi;

```

```

pnl_mat_mult_mat_inplace(interFF, F, F);
j2 *= 2.0 * trace(interFF) - 4.0 * (1.0 - trace(F)) * psi * (1.0 - psi) * vfv
      psi * psi * (9.0 + (2.0 - 3.0 * uu) * psi - uu * (4.0 - uu) * psi * psi)
      2.0 * psi * (5.0 + (1.0 - 2.0 * uu) * psi) * pnl_vect_scalar_prod(d_V, i

//17
I = j0 + j1 - 0.5 * j2;

res -= exp(-r * t) * K * I;

inter2 = -pnl_vect_get(s0, 0);
for (i = 1; i <= N; i++)
{
    inter2 += pnl_vect_get(s0, i);
}

//Price
if (flag_app > 0)
{
    *ptprice = res;
}
else
{
    *ptprice = res + K * exp(-r * t) + inter2;
}

pnl_mat_free(&Sigma11);
pnl_mat_free(&cholSigma11);
pnl_mat_free(&sqrts11);
pnl_mat_free(&d2x0);
pnl_mat_free(&F);
pnl_mat_free(&E);
pnl_mat_free(&interMM);
pnl_mat_free(&interFF);
free(mu);
free(nu);
free(emu);
pnl_vect_free(&dd_V);
pnl_vect_free(&d_V);
pnl_vect_free(&interM1_V);
pnl_vect_free(&interM2_V);

```

```

    pnl_vect_free(&Sigma10_V);
    pnl_vect_free(&s11is10_V);
    pnl_vect_free(&dx0_V);
    pnl_vect_free(&d_last_V);
    pnl_vect_free(&canoni);
}

static int ap_multispread(PnlVect *s0,
                          NumFunc_nd *p,
                          double t,
                          double r,
                          PnlVect *divid,
                          PnlVect *sigma,
                          double rho,
                          double *ptprice,
                          PnlVect *ptdelta)
{
    //int BS_Dimension = s0->size;
    int put_or_call;
    double K = p->Par[0].Val.V_DOUBLE;

    if ((p->Compute) == &CallMultiSpread_nd)
        put_or_call = 1;
    else
        put_or_call = -1;

    ap_spread_li(s0, K, t, r, divid, sigma, ptdelta, rho, put_or_call, ptprice);

    return OK;
}

int CALC(AP_MultiSpread)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r;
    int i, res;
    PnlVect *divid = pnl_vect_create(ptMod->Size.Val.V_PINT);
    PnlVect *spot, *sig;

```

```

spot = ptMod->S0.Val.V_PNLVECT;
sig = ptMod->Sigma.Val.V_PNLVECT;

for (i = 0; i < ptMod->Size.Val.V_PINT; i++)
    pnl_vect_set(divid, i,
        log(1. + GET(ptMod->Divid.Val.V_PNLVECT, i) / 100.));

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

res = ap_multispread(spot,
    ptOpt->PayOff.Val.V_NUMFUNC_ND,
    ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
    r, divid, sig,
    ptMod->Rho.Val.V_DOUBLE,
    &(Met->Res[0].Val.V_DOUBLE), Met->Res[1].Val.V_PNLVECT);
pnl_vect_free(&divid);

return res;
}

static int CHK_OPT(AP_MultiSpread)(void *Opt, void *Mod)
{
    Option *ptOpt = (Option *)Opt;
    //TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);
    if ((strcmp(ptOpt->Name, "CallMultiSpreadEuro_nd") == 0) || (strcmp(ptOpt->Nam
        return OK;

    return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    TYPEOPT *opt = (TYPEOPT *) (Opt->TypeOpt);

    if (Met->init == 0)
    {
        Met->init = 1;
        Met->Res[1].Val.V_PNLVECT = NULL;
    }
}

```

```

/* some initialisation */
if (Met->Res[1].Val.V_PNLVECT == NULL)
    Met->Res[1].Val.V_PNLVECT = pnl_vect_create(opt->Size.Val.V_PINT);
else
    pnl_vect_resize(Met->Res[1].Val.V_PNLVECT, opt->Size.Val.V_PINT);

return OK;

}

PricingMethod MET(AP_MultiSpread) =
{
    "AP_MultiSpread",
    {{ " ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_MultiSpread),
    { {"Price", DOUBLE, {100}, FORBID}, {"Deltas", PNLVECT, {1}, FORBID},
      { " ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_MultiSpread),
    CHK_ok,
    MET(Init)
};

```