

Help

```

#include "hes1d_std.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_root.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2012+2) //The "#els

static int CHK_OPT(AP_CosineBermudan)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(AP_CosineBermudan)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static PnlMatComplex *be;
static int Nnumber, gflag, gM, jdex;
static double glambda, gvbar, geta, gv0, gT, gr, gstrike, grho, ga, gb;

void static gauss_legendre_quadrature_weight(double av, double bv, PnlVect *x, P
{
    PnlVect *x0, *x1, *w1;
    int size = 64, i;
    x0 = pnl_vect_create_from_zero(128);
    x1 = pnl_vect_create_from_list(size, 0.0122236989606157641980521, 0.036663790
    w1 = pnl_vect_create_from_list(size, 0.02444461801962625182113259, 0.0244315690

    for (i = 0; i < size; i++)
    {
        pnl_vect_set(x0, size - i - 1, -GET(x1, i));
        pnl_vect_set(w, size - i - 1, (bv - av) / 2.*GET(w1, i));
        pnl_vect_set(x0, size + i, GET(x1, i));
    }
}

```

```

    pnl_vect_set(w, size + i, (bv - av) / 2.*GET(w1, i));
}
for (i = 0; i < 2 * size; i++)
{
    pnl_vect_set(x, i, av + (bv - av) / 2.*(GET(x0, i) + 1.));
}
pnl_vect_free(&x1);
pnl_vect_free(&w1);
pnl_vect_free(&x0);
}
void static fq(double lambda, double vbar, double eta, double *result)
{
    *result = 2.*lambda * vbar / pow(eta, 2.) - 1.;
}
void static fzeta(double lambda, double eta, double delta, double *result)
{
    *result = 2.*lambda / ((1.0 - exp(-lambda * delta)) * pow(eta, 2.0));
}
void static fdlnv(double sigmat, double sigmas, double t, double s, double lambda)
{
    *result = zeta * exp(-zeta * (exp(sigmas) * exp(-lambda * (t - s)) + exp(sigmat)));
}
static double fdensity(double x, void *p)
{
    double q, zeta, density;
    fq(glambd, gvbar, geta, &q);
    fzeta(glambd, geta, gT, &zeta);
    fdlnv(x, log(gv0), gT, 0.0, glambd, zeta, q, &density);
    return density - 1.0e-7;
}
void static range_lnv(double lambda, double vbar, double eta, double v0, double
{
    double q, zeta, vlogmean, x0, x1, x2, roots, epsabs, epsrel;
    PnlFunc func;
    int N_max;
    double leftb;
    fq(lambda, vbar, eta, &q);
    fzeta(lambda, eta, T, &zeta);
    vlogmean = log(v0 * exp(-lambda * T) + vbar * (1. - exp(-lambda * T)));
    x0 = vlogmean;
    x1 = vlogmean - 20. / (1. + q);

```

```

x2 = vlogmean + 10. / (1. + q);

func.F = fdensity;
func.params = NULL;
epsabs = 1.0e-8;
epsrel = 1.0e-9;
N_max = 50;

pnl_root_bisection(&func, x1, x0, epsrel, epsabs, N_max, &roots);
*av = roots;
leftb = roots;
pnl_root_bisection(&func, leftb + 0.001, x2, epsrel, epsabs, N_max, &roots);
*bv = roots;
}

void static range_x(double r, double lambda, double vbar, double eta, double v0,
{
    double L = 12;
    double c1 = r * T + (1. - exp(-lambda * T)) * (vbar - v0) / (2.0 * lambda) - 0
    double c2 = (1.0 / (8.0 * pow(lambda, 3))) * (eta * lambda * T * exp(-lambda *
/*Truncation range*/
    *ax = c1 - L * pow(fabs(c2), 0.5) + log(S0 / K);
    *bx = c1 + L * pow(fabs(c2), 0.5) + log(S0 / K);
}

void static funcphi(double a, double b, int number, int jpoint, double T, int M,
{
    double q = 0.0, zeta = 0.0, sigmat, sigmas, omega, delta = T / M;
    int n, i, j;
    int inde[3] = {0, 0, 0};
    dcomplex nv = CZERO, gamma = CZERO, egamma = CZERO;
    PnlMatComplex *cbess = pnl_mat_complex_create(jpoint, jpoint);
    PnlMat *bess = pnl_mat_create(jpoint, jpoint);

    inde[0] = 0;
    fq(lambda, vbar, eta, &q);
    fzeta(lambda, eta, delta, &zeta);
    for (j = 0, inde[1] = 0; j < jpoint; j++, inde[1]++)
    {
        for (i = 0, inde[2] = 0; i < j; i++, inde[2]++)
        {
            sigmat = GET(x, i);
            sigmas = GET(x, j);

```

```

        pnl_hmat_complex_set(ttphi, inde, CRmul(CONE, zeta * exp(-zeta * (exp(
    }
for (i = j, inde[2] = j; i < jpoint; i++, inde[2]++)
{
    sigmat = GET(x, i);
    sigmas = GET(x, j);
    pnl_mat_set(bess, i, j, pnl_bessel_i(q, 2.*zeta * exp(-lambda * T / M
    pnl_hmat_complex_set(ttphi, inde, CRmul(CONE, zeta * exp(-zeta * (exp(
}

}

for (n = 1, inde[0] = 1; n < number; n++, inde[0]++)
{
    omega = n * M_PI / (b - a);
    nv = RCadd(omega * (lambda * rho / eta - 1. / 2.), CRmul(CI, 0.5 * omega *
    gamma = Csqrt(RCsub(pow(lambda, 2.), Cmul(RCmul(2.*pow(eta, 2.), nv), CI))
    egamma = Cexp(CRmul(gamma, -T / M));
    for (j = 0, inde[1] = 0; j < jpoint; j++, inde[1]++)
    {
        for (i = 0, inde[2] = 0; i < j; i++, inde[2]++)
        {
            sigmat = GET(x, i);
            sigmas = GET(x, j);
            pnl_hmat_complex_set(ttphi, inde, RCmul(zeta * exp(-zeta * (exp(si
        }
        for (i = 0, inde[2] = 0; i < jpoint; i++, inde[2]++)
        {
            sigmat = GET(x, i);
            sigmas = GET(x, j);
            pnl_mat_complex_set(cbess, i, j, pnl_complex_bessel_i(q, RCmul(sqr

            pnl_hmat_complex_set(ttphi, inde, RCmul(zeta * exp(-zeta * (exp(si
        }
    }

}

pnl_mat_complex_free(&cbess);
pnl_mat_free(&bess);
}

void static fxi(double a, double b, double l, double u, int n, double *result)
{
    *result = (n == 0) ? (exp(u) - exp(l)) : 1. / (1. + pow(n * M_PI / (b - a) , 2

```

```

}
void static fpsi(double a, double b, double l, double u, int n, double *result)
{
    *result = (n == 0) ? (u - l) : (sin(n * M_PI * (u - a) / (b - a)) - sin(n * M_
}
void static fV(double a, double b, double strike, int n, int i, int m, int matur
{
    double xi, psi, l, u, result = 0.;
    if (flag == -1) //put option
    {
        if (m == maturity)
        {
            fxi(a, b, a, 0., n, &xi);
            fpsi(a, b, a, 0., n, &psi);
            result = 2. / (b - a) * flag * strike * (xi - psi);
        }
        else
        {
            u = (MGET(star, m + 1, i) <= 0.) ? MGET(star, m + 1, i) : 0.;
            l = (a <= 0.) ? a : 0.;
            fxi(a, b, l, u, n, &xi);
            fpsi(a, b, l, u, n, &psi);
            result = MGET(chat, n, i) + 2. / (b - a) * flag * strike * (xi - psi);
        }
    }
    else if (flag == 1) // call option
    {
        if (m == maturity)
        {
            fxi(a, b, 0., b, n, &xi);
            fpsi(a, b, 0., b, n, &psi);
            result = 2. / (b - a) * flag * strike * (xi - psi);
        }
        else
        {
            u = (b >= 0.) ? b : 0.;
            l = (MGET(star, m + 1, i) >= 0.) ? MGET(star, m + 1, i) : 0.;
            fxi(a, b, l, u, n, &xi);
            fpsi(a, b, l, u, n, &psi);
            result = MGET(chat, n, i) + 2. / (b - a) * flag * strike * (xi - psi);
        }
    }
}

```

```

    }
    pnl_mat_set(ve, n, i, result);
}
void static funcbeta(int n, int j, int jpoint, PnlVect *w, PnlHmatComplex *ttphi
{
    int i;
    int inde[3] = {0, 0, 0};
    dcomplex sum = CZERO;
    inde[0] = n;
    inde[1] = j;

    for (i = 0; i < jpoint; i++)
    {
        inde[2] = i;
        sum = Cadd(sum, RCmul(pnl_vect_get(w, i) * pnl_mat_get(v, n, i), pnl_hmat_
    }
    pnl_mat_complex_set(be, n, j, sum);
}

void static mbasic(double a, double b, double l, double u, PnlVectComplex *mj)
{
    int n;
    pnl_vect_complex_set(mj, 0, RCmul((u - l)*M_PI / (b - a), CI));
    for (n = 1; n < 2 * Nnumber + 1; n++)
    {
        pnl_vect_complex_set(mj, n, CRdiv(Csub(Cexp(RCmul(n * (u - a)*M_PI / (b -
    }
}

void static c_fft(int j, int number, double T, double M, double a, double b, dou
{
    PnlVectComplex *ms = pnl_vect_complex_create(2 * number);
    PnlVectComplex *mc = pnl_vect_complex_create(2 * number);
    PnlVectComplex *us = pnl_vect_complex_create(2 * number);
    PnlVectComplex *ivector1 = pnl_vect_complex_create(2 * number);
    PnlVectComplex *ivector2 = pnl_vect_complex_create(2 * number);
    int n;

    for (n = 0; n < number; n++)
    {
        pnl_vect_complex_set(ms, n, RCmul(-1, Conj(pnl_vect_complex_get(mj, n))));
    }
}

```

```

        pnl_vect_complex_set(us, n, pnl_mat_complex_get(beta, n, j));
        pnl_vect_complex_set(mc, n, pnl_vect_complex_get(mj, 2 * number - 1 - n));
    }
    pnl_vect_complex_set(us, 0, RCmul(0.5, pnl_mat_complex_get(beta, 0, j)));
    for (n = Nnumber; n < 2 * number; n++)
    {
        pnl_vect_complex_set(ms, n, pnl_vect_complex_get(mj, 2 * number - n));
        pnl_vect_complex_set(us, n, CZERO);
        pnl_vect_complex_set(mc, n, pnl_vect_complex_get(mj, 2 * number - 1 - n));
    }
    pnl_vect_complex_set(ms, number, CZERO);

    pnl_fft_inplace(us);
    pnl_fft_inplace(ms);
    pnl_fft_inplace(mc);

    for (n = 0; n < 2 * number; n++)
    {
        pnl_vect_complex_set(ivector1, n, Cmul(pnl_vect_complex_get(ms, n), pnl_vect_complex_get(us, n)));
        pnl_vect_complex_set(ivector2, n, Cmul(pnl_vect_complex_get(mc, n), pnl_vect_complex_get(us, n)));
    }
    for (n = 0; n < number; n++)
    {
        pnl_vect_complex_set(ivector2, 2 * n + 1, RCmul(-1., pnl_vect_complex_get(ivector1, n)));
    }
    pnl_ifft_inplace(ivector1);
    pnl_ifft_inplace(ivector2);
    for (n = 0; n < number; n++)
    {
        pnl_mat_set(chat, n, j, exp(-r * T / M) / M_PI * (Cimag(pnl_vect_complex_get(ivector2, 2 * n + 1))));
    }

    pnl_vect_complex_free(&ms);
    pnl_vect_complex_free(&mc);
    pnl_vect_complex_free(&us);
    pnl_vect_complex_free(&ivector1);
    pnl_vect_complex_free(&ivector2);
}

static void fdf(double x, double *f, double *df, void *p)
{

```

```

int n;
dcomplex sum = CZERO, sumd = CZERO;
double g = 0.0, dg = 0.0;
sum = RCmul(0.5, pnl_mat_complex_get(be, 0, jdex));
sumd = CZERO;
for (n = 1; n < Nnumber; n++)
{
    sum = Cadd(sum, Cmul(pnl_mat_complex_get(be, n, jdex), Cexp(CRmul(CI, n *
    sumd = Cadd(sumd, Cmul(pnl_mat_complex_get(be, n, jdex), Cmul(Cexp(CRmul(C
    }
    g = (gflag * (exp(x) - 1.0) >= 0.0) ? gflag * (exp(x) - 1.0) : 0.0;
    *f = exp(-gr * gT / gM) * Creal(sum) - gstrike * g ;
    dg = (gflag * (exp(x) - 1.0) >= 0.0) ? gflag * exp(x) : 0.0;
    *df = exp(-gr * gT / gM) * Creal(sumd) - gstrike * dg;
    if (fabs(exp(-gr * gT / gM)*Creal(sumd) - gstrike * dg) < 1E-9)
    {
        printf("derivative is zero!!    sumd=%f+%fi, dg=%f, %f-%f=%f, df=%f\ n", sum
    }
}

void static newton_root_find(int m, int j, PnlMat *star)
{
    double x0 = log(1.), r;
    PnlFuncDFunc func;
    static double epsabs = 0.0000001;
    static double epsrel = 0.00000001;
    static int N_max = 10;

    if (m == gM)        x0 = log(1.);
    else    x0 = MGET(star, m + 1, j);
    func.F = fdf;
    func.params = NULL;

    pnl_root_newton(&func, x0, epsrel, epsabs, N_max, &r);
    MLET(star, m, j) = r;
}

/*double fdf(double x, void *p)
{
    int n;
    dcomplex sum=CZERO;
    double g=0.0;

```



```

        sum= RCmul(0.5, pnl_mat_complex_get(be, 0, jdex));
        for(n=1; n<Nnumber; n++)
        {
            sum= Cadd(sum, Cmul(pnl_mat_complex_get(be, n, jdex), Cexp(CRmul
        }
        g=(gflag*(exp(x)-1.0)>=0.0 )? gflag*(exp(x)-1.0): 0.0;
//      printf("x=%f, c=%f, g=%f, c-g=%f \ n", x*100., exp(-gr*gT/gM)*Creal(
        return exp(-gr*gT/gM)*Creal(sum)-gstrike*g;
    }
void static bisection_root_find(int m, int j)
{
    double x1, x2, r=0.0;
    PnlFunc func;
    int status;

    x1 = ga;
    x2 = gb;
    func.F = fdf;
    func.params = NULL;
    static double epsabs = 0.000001;
    static double epsrel = 0.000001;
    static int N_max = 1000;
    printf("x1=%f,x2=%f    ",x1, x2);

    if (m==gM) status = pnl_root_bisection(&func, x1, 0.00001, epsrel, epsabs, N_
    else status = pnl_root_bisection(&func, x1, MGET(star, m+1, j), epsrel, epsab
    printf("m=%d, j=%d, bisection : root is %f, status==OK %d\ n", m, j, r, statu
    MLET(star, m, j) = r;
}
*/
void static Bermuda_Heston(double S0, double strike, double T, double r, double
{
    dcomplex suml = CZERO, sumr = CZERO, sum = CZERO;
    int m, i, j, n;
    double av, bv, ax, bx, c3 = 0.0, cl, cr, lnS0, miu;
    int dims[3] = {number, jpoint, jpoint};
    int index[3] = {0, 0, 0};
    PnlHmatComplex *tphi = pnl_hmat_complex_create(3, dims);
    PnlVect *x = pnl_vect_create_from_zero(jpoint);
    PnlVect *w = pnl_vect_create_from_zero(jpoint);
    PnlVectComplex *mj = pnl_vect_complex_create(2 * number + 1);

```

```

PnlMat *star = pnl_mat_create(M + 1, jpoint);
PnlMat *chat = pnl_mat_create(number, jpoint);
PnlMat *ve = pnl_mat_create(number, jpoint);

be = pnl_mat_complex_create(number, jpoint);

range_lnv(lambda, vbar, eta, v0, T, &av, &bv);
range_x(r, lambda, vbar, eta, v0, T, rho, S0, strike, &ax, &bx);

gauss_legendre_quadrature_weight(av, bv, x, w);

lnS0 = log(S0 / strike);
miu = r - dividend;
ga = ax;
gb = bx;
for (n = 0; n < number; n++)
{
    fV(ga, gb, strike, n, 0, M, M, flag, chat, star, ve);
    for (i = 1; i < jpoint; i++)
    {
        pnl_mat_set(ve, n, i, pnl_mat_get(ve, n, 0));
    }
}
funcphi(ax, bx, number, jpoint, T, M, lambda, rho, eta, vbar, miu, x, tphi);
for (n = 0, index[0] = 0; n < number; n++, index[0]++)
{
    for (j = 0, index[1] = 0; j < jpoint; j++, index[1]++)
    {
        funcbeta(n, j, jpoint, w, tphi, ve);
    }
}

for (j = 0; j < jpoint; j++)
{
    jdex = j;
    newton_root_find(M, j, star); // bisection_root_find(m, j); //
    mbasic(ax, bx, MGET(star, M, j), bx, mj);
    c_fft(j, number, T, M, ax, bx, MGET(star, M, j), bx, r, mj, be, chat);
}
for (m = M - 1; m > 1; m--)
{

```

```

for (n = 0; n < number; n++)
{
    for (i = 0; i < jpoint; i++)
    {
        fV(ax, bx, strike, n, i, m, M, flag, chat, star, ve);
    }
}
for (n = 0, index[0] = 0; n < number; n++, index[0]++)
{
    for (j = 0, index[1] = 0; j < jpoint; j++, index[1]++)
    {
        funcbeta(n, j, jpoint, w, tphi, ve);
    }
}
for (j = 0; j < jpoint; j++)
{
    jdex = j;
    newton_root_find(m, j, star); // bisection_root_find(m, j); //
    mbasic(ax, bx, MGET(star, m, j), bx, mj);
    c_fft(j, number, T, M, ax, bx, MGET(star, m, j) , bx, r, mj, be, chat)
}
}

if (log(v0) >= pnl_vect_get(x, jpoint - 1))
{
    for (i = 0; i < jpoint; i++)
    {
        fV(ax, bx, strike, 0, i, 1, M, flag, chat, star, ve);
    }
    funcbeta(0, jpoint - 1, jpoint, w, tphi, ve);
    sum = RCmul(0.5, Cmul(pnl_mat_complex_get(be, 0, jpoint - 1), CONE));
    for (n = 1; n < number; n++)
    {
        for (i = 0; i < jpoint; i++)
        {
            fV(ax, bx, strike, n, i, 1, M, flag, chat, star, ve);
        }
        funcbeta(n, jpoint - 1, jpoint, w, tphi, ve);
        sum = Cadd(sum, Cmul(pnl_mat_complex_get(be, n, jpoint - 1), Cexp(CRmu
    }
}
c3 = exp(-r * T / M) * Creal(sum);

```

```

    }
else
{
    for (j = 0; j < jpoint - 1; j++)
    {
        if (log(v0) == pnl_vect_get(x, j))
        {
            for (i = 0; i < jpoint; i++)
            {
                fV(ax, bx, strike, 0, i, 1, M, flag, chat, star, ve);
            }
            funcbeta(0, j, jpoint, w, tphi, ve);
            sum = RCmul(0.5, Cmul(pnl_mat_complex_get(be, 0, j), CONE));

            for (n = 1; n < number; n++)
            {
                for (i = 0; i < jpoint; i++)
                {
                    fV(ax, bx, strike, n, i, 1, M, flag, chat, star, ve);
                }
                funcbeta(n, j, jpoint, w, tphi, ve);
                sum = Cadd(sum, Cmul(pnl_mat_complex_get(be, n, j), Cexp(CRmul
            }
            c3 = exp(-r * T / M) * Creal(sum);
        }
        else if (log(v0) < pnl_vect_get(x, j + 1) && log(v0) > pnl_vect_get(x,
        {
            for (i = 0; i < jpoint; i++)
            {
                fV(ax, bx, strike, 0, i, 1, M, flag, chat, star, ve);
            }
            funcbeta(0, j, jpoint, w, tphi, ve);
            funcbeta(0, j + 1, jpoint, w, tphi, ve);
            suml = RCmul(0.5, Cmul(pnl_mat_complex_get(be, 0, j), CONE));
            sumr = RCmul(0.5, Cmul(pnl_mat_complex_get(be, 0, j + 1), CONE));

            for (n = 1; n < number; n++)
            {
                for (i = 0; i < jpoint; i++)
                {
                    fV(ax, bx, strike, n, i, 1, M, flag, chat, star, ve);

```

```

        }
        funcbeta(n, j, jpoint, w, tphi, ve);
        funcbeta(n, j + 1, jpoint, w, tphi, ve);
        suml = Cadd(suml, Cmul(pnl_mat_complex_get(be, n, j) , Cexp(C
        sumr = Cadd(sumr, Cmul(pnl_mat_complex_get(be, n, j + 1) , Ce
    }
    cl = exp(-r * T / M) * suml.r;
    cr = exp(-r * T / M) * sumr.r;
    c3 = cl + (log(v0) - pnl_vect_get(x, j)) / (pnl_vect_get(x, j + 1)
}
}
}
*price = c3;
pnl_hmat_complex_free(&tphi);
pnl_vect_free(&x);
pnl_vect_free(&w);
pnl_vect_complex_free(&mj);
pnl_mat_free(&chat);
pnl_mat_free(&ve);
pnl_mat_free(&star);
}

```

```

static int Cosine_Bermudan(double S0, double strike, double T, double r, double
{
    int N, J;

    //Fixed Parameters. Don't modify.
    N = (int)pow(2, 7);
    J = (int)pow(2, 7);

    gflag = iscall;
    gstrike = strike;
    gT = T;
    gr = r;
    glambda = lambda;
    geta = eta;
    gvbar = vbar;
    gv0 = v0;
    grho = rho;
    gM = M;

```

```

Nnumber = N;

Bermuda_Heston(S0, strike, T, r, q, lambda, eta, vbar, v0, rho, M, iscall, N,

pnl_mat_complex_free(&be);

return OK;
}

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

    iscall = -1; //Put Case
    if (ptOpt->PayOff.Val.V_NUMFUNC_1->Compute == &Call) iscall = 1;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
    Met->Res[1].Val.V_DOUBLE = 0.;

    return Cosine_Bermudan(ptMod->S0.Val.V_PDOUBLE,
                           ptOpt->PayOff.Val.V_NUMFUNC_1->Par[0].Val.V_PDOUBLE,
                           ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, r, di
                           ptMod->Sigma0.Val.V_PDOUBLE,
                           ptMod->LongRunVariance.Val.V_PDOUBLE,
                           ptMod->MeanReversion.Val.V_PDOUBLE,
                           ptMod->Sigma.Val.V_PDOUBLE,
                           ptMod->Rho.Val.V_PDOUBLE,
                           iscall,
                           Met->Par[0].Val.V_INT,
                           &(Met->Res[0].Val.V_DOUBLE));
}

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

```

```

        return OK;

    return  WRONG;
}

#endif

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->Par[0].Val.V_INT = 20;
        Met->init = 1;
        Met->HelpFilenameHint = "ap_cosine_bermhes1d";
    }
    return OK;
}

PricingMethod MET(AP_CosineBermudan) =
{
    "AP_CosineBermudan",
    {"Bermudan Steps", INT, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_CosineBermudan),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_CosineBermudan),
    CHK_ok,
    MET(Init)
};

```