

## Help

```
#include "bergomirev2d_std.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_list.h"
#include "pnl/pnl_integration.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_basis.h"

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

static PnlMat *ResBergomi;

static double HermitePoly(int n, double x)
{
    switch (n)
    {
        {
            case 0:
                return 1;
                break;
            case 1:
                return x;
                break;

            case 2 :
                return SQR(x) - 1;
                break;

            case 3:

```

```

        return POW(x, 3) - 3 * x;
        break;

    case 4:
        return POW(x, 4) - 6 * POW(x, 2) + 3;
        break;
    case 5:

        return POW(x, 5) - 10 * POW(x, 3) + 15 * x;
        break;
    case 6:
        return POW(x, 6) - 15 * POW(x, 4) + 45 * POW(x, 2) - 15 ;
        break;
    default:
        return 0;
        break;
    }
}

//-----A-----
static double a_1(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, d
{
    double coeff, res = 0.0;
    int j;

    if (i >= T->size)
        return 0.0;
    else
    {
        coeff = pnl_vect_get(m, i) * (exp(-2.0 * k * pnl_vect_get(T, i)) - exp(-2.
        for (j = 0; j < i; j++)
            res += SQR(pnl_vect_get(omega, j) * theta) * exp(-2.0 * k * pnl_vect_get

        return coeff * res;
    }
}

static double a_2(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, d
{
    return pnl_vect_get(m, i) * SQR(pnl_vect_get(omega, i) * theta) * SQR(1.0 - e
}

```

```

static double An(PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k)
{
    int i;
    double res = 0.0;

    int M;
    M = T->size - 1;

    for (i = 0; i < M; i++)
        res += (
            a_1(i, T, m, omega, rho, k, theta) +
            a_2(i, T, m, omega, rho, k, theta)
        );
    return res;
}

//-----B-----
static double b_1(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k)
{
    double coeff, res = 0.0;
    int j;

    if (i >= T->size)
        return 0.0;
    else
    {
        coeff = pnl_vect_get(m, i) * (exp(-k * pnl_vect_get(T, i)) - exp(-k * pnl_vect_get(T, i))) / 2.0;
        for (j = 0; j < i; j++)
        {
            res -= SQR(pnl_vect_get(omega, j) * theta) / 2.0 * exp(-2.0 * k * pnl_vect_get(T, j));
        }
        return coeff * res;
    }
}

static double b_2(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k)
{
    return (
        - pnl_vect_get(m, i) * SQR(pnl_vect_get(omega, i) * theta) / 2.0 * (

```

```

    );

}

static double Bn(PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k)
{
    int i;
    double res = 0.0;

    int M;
    M = T->size - 1;
    for (i = 0; i < M; i++)
        res += (
            b_1(i, T, m, omega, rho, k, theta) +
            b_2(i, T, m, omega, rho, k, theta)
        );

    return res;
}

//-----C-----
static double c_1(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k)
{
    double coeff, res = 0.0;
    int j;

    if (i >= T->size)
        return 0.0;

    coeff = pnl_vect_get(m, i) * (exp(-k * pnl_vect_get(T, i)) - exp(-k * pnl_vect_get(T, i)))
    for (j = 0; j < i; j++)
        res = pnl_vect_get(rho, j) * sqrt(pnl_vect_get(m, j)) * pnl_vect_get(omega, j);

    return coeff * res;
}

static double c_2(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k)
{
    return (
        pnl_vect_get(m, i) * pnl_vect_get(rho, i) * sqrt(pnl_vect_get(m, i))
    );
}

```

```

}
```

```

static double Cn(PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k
{
    int i;
    double res = 0.0;

    int M;
    M = T->size - 1;

    for (i = 0; i < M; i++)
        res += (
            c_1(i, T, m, omega, rho, k, theta) +
            c_2(i, T, m, omega, rho, k, theta)
        );

    return res;
}
```

```

//-----D-----
static double d_1(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho ,
{
    double res = 0.0;
    int j;

    if (i >= T->size)
        return 0.0;
    else
    {
        for (j = 0; j < i; j++)
            res = pnl_vect_get(rho, j) * sqrt(pnl_vect_get(m, j)) * pnl_vect_get(om

        return res * b_1(i, T, m, omega, rho, k_m, theta_m);
    }
}
```

```

static double d_2(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho ,
{
    double res = 0.0;
    int j;
```

```

    if (i >= T->size)
        return 0.0;
    else
    {
        for (j = 0; j < i; j++)
            res = pnl_vect_get(rho, j) * sqrt(pnl_vect_get(m, j)) * pnl_vect_get(om

        return res * b_2(i, T, m, omega, rho, k_m, theta_m);
    }
}

static double d_3(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho ,
{
    double coeff, res = 0.0, delta;
    int j;

    if (i >= T->size)
        return 0.0;
    else
    {
        delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

        coeff = pnl_vect_get(m, i) * sqrt(pnl_vect_get(m, i)) * pnl_vect_get(rho,
            (
                (exp(-k_m * pnl_vect_get(T, i + 1)) - exp(-k_n * 0.9999999 * del
            -
                (exp(-k_n * pnl_vect_get(T, i + 1)) - exp(-k_n * delta - k_n * p
            ));

        for (j = 0; j < i; j++)
            res -= SQR(pnl_vect_get(omega, j) * theta_m) / 2.0 * exp(-2.0 * k_m * pn

        return coeff * res;
    }
}

static double d_4(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, d
{
    double delta;
    double k;

```

```

k = k_n * 0.99999999;

if (i >= T->size)
    return 0.0;
else
{
    delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

    return (-0.5 * CUB(sqrt(pnl_vect_get(m, i))) * pnl_vect_get(rho, i) * CUB(
        (
            (1 - exp(-(k + k_m) * delta)) / (k + k_m) -
            (exp(-2.0 * k * delta) - k / k_m * exp(-(k + k_m) * delta)) / (k
            -
            exp(-k * delta) / k_m - delta * exp(k_m * pnl_vect_get(T, i) - (
            )
            )
        )
    );
}
}

static double Dn(PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k_n,
{
    int i;
    double res = 0.0;

    int M;
    M = T->size - 1;

    for (i = 0; i < M; i++)
        res += (
            d_1(i, T, m, omega, rho, k_n, theta_n, k_m, theta_m) +
            d_2(i, T, m, omega, rho, k_n, theta_n, k_m, theta_m) +
            d_3(i, T, m, omega, rho, k_n, theta_n, k_m, theta_m) +
            d_4(i, T, m, omega, rho, k_n, theta_n, k_m, theta_m)
        );

    return res;
}

```

```

////////////////////////////////////

```

```

////////// EEEEEEEEEEEEEEEEEEE
////////////////////////////////////

static double E_i_j(int i, int j, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect
{

    int l;
    double res = 0.0;


    if (j < i || j >= T->size)
        return 0.0;
    else
    {
        for (l = i; l < j; l++)
            res += SQR(theta) * SQR(pnl_vect_get(omega, l)) * exp(-2.0 * k * pnl_vect_

        res *= pnl_vect_get(m, j) * (exp(-k * pnl_vect_get(T, j)) - exp(-k * pnl_v

        res += pnl_vect_get(m, j) * (exp(-k * pnl_vect_get(T, i)) - exp(-k * pnl_v
            (
                (exp(3.0 * k * pnl_vect_get(T, j + 1)) - exp(3.0 * k * pnl_vect_g
                exp(4.0 * k * pnl_vect_get(T, j)) * (exp(-k * pnl_vect_get(T, j))
            );
        }
    return res;
}

static double tildeE_i_j(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect
{

    int l;
    double res = 0.0;


    if (i >= T->size)
        return 0.0;
    else
    {

```



```

        for (l = i; l < i; l++)
            res += SQR(theta) * SQR(pnl_vect_get(omega, l)) * exp(-2.0 * k * pnl_vect_get(m, i)) * ((exp(-2.0 * k * pnl_vect_get(T, i)) - exp(-2.0 * k * pnl_vect_get(T, i + 1))) * exp(-2.0 * k * pnl_vect_get(T, i + 1)));

        res += pnl_vect_get(m, i) * SQR(theta) * SQR(pnl_vect_get(omega, i)) * exp(-2.0 * k * pnl_vect_get(T, i)) * ((exp(3.0 * k * pnl_vect_get(T, i + 1)) - exp(3.0 * k * pnl_vect_get(T, i))) * exp(3.0 * k * pnl_vect_get(T, i)) * (exp(-k * pnl_vect_get(T, i)) * exp(4.0 * k * pnl_vect_get(T, i)) * (exp(-2.0 * k * pnl_vect_get(T, i)) - exp(-2.0 * k * pnl_vect_get(T, i + 1))) * exp(-2.0 * k * pnl_vect_get(T, i + 1))) * exp(-2.0 * k * pnl_vect_get(T, i + 1)));

        return res;
    }
}

static double En(PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k,
{
    double res = 0.0, aux = 0.0;
    int i, j;
    int M;
    M = T->size - 1;

    for (i = 0; i < M; i++)
    {
        aux = 0.0;
        for (j = 0; j < i; j++)
            aux += E_i_j(i, j, T, m, omega, rho, k, theta);

        res += pnl_vect_get(m, i) * (aux + tildeE_i_j(i, T, m, omega, rho, k, theta));
    }
    return res;
}

////////////////////////////////////
////////////////////////////////////
//////////////////////////////////// FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
////////////////////////////////////

static double f_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho,
{

```

```

    int j;
    double res = 0.0;
    if (i >= T->size)
        return 0.0;
    else
    {
        for (j = 0; j < i; j++)
            res += sqrt(pnl_vect_get(m, j)) * pnl_vect_get(omega, j) * pnl_vect_get(
            return res;
        }
    }

static double tildef_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect
{
    double k, Delta;

    k = k_n * 0.99999999;

    if (i >= T->size)
        return 0.0;
    else
    {
        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i) ;

        return (pnl_vect_get(m, i) * sqrt(pnl_vect_get(m, i)) * pnl_vect_get(omega
            (
                exp(- k_m * pnl_vect_get(T, i + 1)) / (2.0 * k * (k - k_m)) -
                exp(-k * Delta - k_m * pnl_vect_get(T, i)) / (k * k - k_m * k_m)
                exp(-2.0 * k * Delta - k_m * pnl_vect_get(T, i + 1)) / (2.0 * (k
            ));
    }
}

static double F1_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *r
{
    if (i >= T->size)
        return 0.0;

```

```

else
{
    return (
        pnl_vect_get(m, i) * (exp(- (k_n + k_m) * pnl_vect_get(T, i)) - e
        f_i_n(i, T, m, omega, rho_n, k_n , theta_n) * f_i_n(i, T, m,
        );
    }
}

static double F2_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *
{
    if (i >= T->size)
        return 0.0;
    else
    {
        return (
            tildef_i_n_m(i, T, m, omega, rho_n, k_n , theta_n, k_m , theta_m)
        );
    }
}

static double F3_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *
{
    return F2_i_n_m(i, T, m, omega, rho_m, rho_n, k_m, theta_m, k_n, theta_n);
}

static double F4_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *
{
    double k, Delta;
    if (i >= T->size)
        return 0.0;
    else
    {
        k = k_n * 0.9999999;
        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

        return (
            SQR(pnl_vect_get(m, i)) * SQR(pnl_vect_get(omega, i)) * pnl_vect_
            (
                (1 - exp(-2.0 * (k + k_m) * Delta)) / (k + k_m) + (exp(-2.0 * k
            )

```

```

        );
    }
}

```

```

static double F_n_m(PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho_n, PnlVect *rho_m, PnlVect *k_n, PnlVect *k_m, PnlVect *theta_n, PnlVect *theta_m)
{
    int i;
    double res = 0.0;

    int M;
    M = T->size - 1;

    for (i = 0; i < M; i++)
        res += (
            F1_i_n_m(i, T, m, omega, rho_n, rho_m, k_n, theta_n, k_m, theta_m)
            F3_i_n_m(i, T, m, omega, rho_n, rho_m, k_n, theta_n, k_m, theta_m)
        );

    return res;
}

```

```

////////////////////
////////////////////
//////// GGGGGGGGGG
////////////////////

```

```

static double g1_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho_n, PnlVect *rho_m, PnlVect *k_n, PnlVect *k_m, PnlVect *theta_n, PnlVect *theta_m)
{
    int j;
    double res = 0.0;
    if (i >= T->size)
        return 0.0;
    else
    {
        for (j = 0; j < i; j++)
            res += sqrt(pnl_vect_get(m, j)) * pnl_vect_get(omega, j) * pnl_vect_get(rho_n, j) * pnl_vect_get(rho_m, j) * pnl_vect_get(k_n, j) * pnl_vect_get(k_m, j) * pnl_vect_get(theta_n, j) * pnl_vect_get(theta_m, j);

        return res;
    }
}

```

```

}
static double g2_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho)
{
    int j;
    double res = 0.0;
    if (i >= T->size)
        return 0.0;
    else
    {
        for (j = 0; j < i; j++)
            res += sqrt(pnl_vect_get(m, j)) * pnl_vect_get(omega, j) * pnl_vect_get(rho, j);

        return res;
    }
}

static double g3_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho)
{
    double res, Delta;
    if (i >= T->size)
        return 0.0;
    else
    {
        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

        res = sqrt(pnl_vect_get(m, i)) * pnl_vect_get(omega, i) * pnl_vect_get(rho, i) *
            (Delta - (1 - exp(-k * Delta)) / k);

        return res;
    }
}

static double g4_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho)
{
    double res, Delta;
    if (i >= T->size)
        return 0.0;
    else

```

```

    {
        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

        res = sqrt(pnl_vect_get(m, i)) * pnl_vect_get(omega, i) * pnl_vect_get(rh
            SQR((1 - exp(-k * Delta)) / k);

        return res;
    }
}

```

```

static double tildeg_i_j_n(int i, int j, PnlVect *T, PnlVect *m, PnlVect *omega,
{
    double res , Delta;
    if (i >= T->size || j >= T->size)
        return 0.0;
    else
    {
        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

        res = g2_i_n(j, T, m, omega, rho, k , theta) * exp(-k * pnl_vect_get(T, j
            (exp(2.0 * k * pnl_vect_get(T, j + 1)) - exp(2.0 * k * pnl_vect_get(

        res += sqrt(pnl_vect_get(m, i)) * pnl_vect_get(omega, i) * pnl_vect_get(r
            exp(-k * pnl_vect_get(T, i + 1)) * (exp(2.0 * k * pnl_vect_get(T,

        return res;
    }
}

```

```

static double G1_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho
{
    if (i >= T->size)
        return 0.0;
    else
    {
        return (

```

```

        pnl_vect_get(m, i) * (exp(-k * pnl_vect_get(T, i)) - exp(-k * pnl_vect_get(T, i + 1))) - exp(-k * pnl_vect_get(T, i)) *
        g1_i_n(i, T, m, omega, rho, k, theta) * g2_i_n(i, T, m, omega, rho, k, theta);
    }
}

static double G2_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho)
{
    return (
        pnl_vect_get(m, i) * g1_i_n(i, T, m, omega, rho, k, theta) * g3_i_n(i, T, m, omega, rho, k, theta);
    );
}

static double G3_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho)
{
    return (
        pnl_vect_get(m, i) * g2_i_n(i, T, m, omega, rho, k, theta) * g4_i_n(i, T, m, omega, rho, k, theta);
    );
}

static double G4_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho)
{
    double Delta;
    if (i >= T->size)
        return 0.0;
    else
    {
        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

        return (
            SQR(pnl_vect_get(m, i)) * SQR(pnl_vect_get(omega, i)) * SQR(theta) *
            ((1 - 2.0 * exp(-k * Delta) + 3.0 * exp(-2.0 * k * Delta) - 2.0 * exp(-3.0 * k * Delta)));
        );
    }
}

static double tildeG1_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho)
{
    int j;
    double res = 0.0;

```

```

    if (i >= T->size)
        return 0.0;
    else
    {
        for (j = 0; j < i; j++)
            res += pnl_vect_get(m, i) * sqrt(pnl_vect_get(m, j)) * pnl_vect_get(rho, j);

        return res;
    }
}

static double tildeG2_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho)
{
    int j;
    double Delta;
    double res = 0.0;
    if (i >= T->size)
        return 0.0;
    else
    {
        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

        for (j = 0; j < i; j++)
            res += pnl_vect_get(m, i) * g4_i_n(i, T, m, omega, rho, j, Delta) * g2_i_n(i, T, m, omega, rho, j, Delta);

        res += SQR(pnl_vect_get(m, i)) * SQR(pnl_vect_get(rho, i)) * SQR(pnl_vect_get(T, i));

        return res;
    }
}

static double Gn(PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho, double k)
{
    int i;
    double res = 0.0;

    int M;
    M = T->size - 1;

    for (i = 0; i < M; i++)
        res += (

```



```

        G1_i_n(i, T, m, omega, rho, k , theta) + G2_i_n(i, T, m, omega, rho
    );

    return res;
}

////////////////////////////////////
////////////////////////////////////
//////////////////////////////////// HHHHHHHHHHHHHHHH
////////////////////////////////////

static double h_i_n(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho,
{
    int j;
    double res = 0.0;
    if (i >= T->size)
        return 0.0;
    else
    {
        for (j = 0; j < i; j++)
            res += -0.5 * SQR(pnl_vect_get(omega, j)) * SQR(theta) * exp(-2.0 * k *

        return res;
    }
}

static double tildeh_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVec
{

    double k, Delta;

    k = k_n * 0.9999999;

    if (i >= T->size)
        return 0.0;
    else
    {

        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i) ;

```

```

        return (-0.5 * pnl_vect_get(m, i) * SQR(pnl_vect_get(omega, i)) * SQR(thet
            (
                exp(- k_m * pnl_vect_get(T, i + 1)) / (3.0 * k * (2.0 * k - k_m)
                exp(-2.0 * k * Delta - k_m * pnl_vect_get(T, i)) / ((2.0 * k - k
                exp(-3.0 * k * Delta - k_m * pnl_vect_get(T, i + 1)) / (3.0 * (k
            ));
    }

}

static double H1_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *r
{
    if (i >= T->size)
        return 0.0;
    else
    {
        return (
            pnl_vect_get(m, i) * (exp(- 2.0 * (k_n + k_m) * pnl_vect_get(T, i
            f_i_n(i, T, m, omega, rho_n, k_n , theta_n) * f_i_n(i, T, m,
        )
    }
}

static double H2_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *
{
    if (i >= T->size)
        return 0.0;
    else
    {
        return (
            tildeh_i_n_m(i, T, m, omega, rho_n, k_n , theta_n, k_m , theta_m)
        );
    }
}

static double H3_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *
{
    return H2_i_n_m(i, T, m, omega, rho_m, rho_n, k_m, theta_m, k_n, theta_n);
}

```

```

static double H4_i_n_m(int i, PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho_n, PnlVect *rho_m, PnlVect *k_n, PnlVect *k_m, PnlVect *theta_n, PnlVect *theta_m)
{
    double k, Delta;
    if (i >= T->size)
        return 0.0;
    else
    {
        k = k_n * 0.99999999;
        Delta = pnl_vect_get(T, i + 1) - pnl_vect_get(T, i);

        return (
            pnl_vect_get(m, i) * pow(pnl_vect_get(omega, i), 4) * SQR(theta_n) *
            (
                (1 - exp(-2.0 * (k + k_m) * Delta)) / (2.0 * (k + k_m)) - (exp(-3.0 * k * Delta) - exp(-2.0 * (k + k_m) * Delta)) / (2.0 * (k + k_m))
            )
        );
    }
}

```

```

static double H_n_m(PnlVect *T, PnlVect *m, PnlVect *omega, PnlVect *rho_n, PnlVect *rho_m, PnlVect *k_n, PnlVect *k_m, PnlVect *theta_n, PnlVect *theta_m)
{
    int i;
    double res = 0.0;

    int M;
    M = T->size - 1;

    for (i = 0; i < M; i++)
        res += (
            H1_i_n_m(i, T, m, omega, rho_n, rho_m, k_n, theta_n, k_m, theta_m) +
            H3_i_n_m(i, T, m, omega, rho_n, rho_m, k_n, theta_n, k_m, theta_m)
        );

    return res;
}

```

```

static double mu1(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *
{
    PnlVect *rho_n, *rho_m;
    int i, n, l, M, N;
    double res = 0.0;

    N = theta->size;
    M = T->size - 1;

    rho_n = pnl_vect_create(M);
    rho_m = pnl_vect_create(M);

    for (i = 0; i < M; i++)
        res += 0.5 * pnl_vect_get(m, i) * (pnl_vect_get(T, i + 1) - pnl_vect_get(T,

    for (n = 0; n < N; n++)
    {
        for (i = 0; i < M; i++)
            pnl_vect_set(rho_n, i, pnl_mat_get(rho, i, n));

        res += 0.25 * An(T, m, omega, rho_n, pnl_vect_get(k, n) , pnl_vect_get(the
        for (l = 0; l < N; l++)
        {
            for (i = 0; i < M; i++)
                pnl_vect_set(rho_m, i, pnl_mat_get(rho, i, l));
            res += 0.25 * H_n_m(T, m, omega, rho_n, rho_m, pnl_vect_get(k, n), pnl
        }
    }
    pnl_vect_free(&rho_n);
    pnl_vect_free(&rho_m);

    return res;

}

static double mu2(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *
{
    PnlVect *rho_n, *rho_m;
    int i, n, l, M, N;
    double res = 0.0;

    N = theta->size;

```

```

M = T->size - 1;
rho_n = pnl_vect_create(M);
rho_m = pnl_vect_create(M);

res = mul(T, m, omega, rho, k , theta);

for (n = 0; n < N; n++)
{
    for (i = 0; i < M; i++)
        pnl_vect_set(rho_n, i, pnl_mat_get(rho, i, n));

    res += -0.5 * Cn(T, m, omega, rho_n, pnl_vect_get(k, n) , pnl_vect_get(theta, n));
    for (l = 0; l < N; l++)
    {
        for (i = 0; i < M; i++)
            pnl_vect_set(rho_m, i, pnl_mat_get(rho, i, l));
        res += 0.5 * Dn(T, m, omega, rho_n, pnl_vect_get(k, n), pnl_vect_get(theta, l));
    }

}

pnl_vect_free(&rho_n);
pnl_vect_free(&rho_m);

return res;

}

static double mu3(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *theta)
{
    PnlVect *rho_n, *rho_m;
    int i, n, l, M, N;
    double res = 0.0;

    N = theta->size;
    M = T->size - 1;
    rho_n = pnl_vect_create(M);
    rho_m = pnl_vect_create(M);

    for (n = 0; n < N; n++)
    {

```

```

        for (i = 0; i < M; i++)
            pnl_vect_set(rho_n, i, pnl_mat_get(rho, i, n));

        res += -0.5 * Cn(T, m, omega, rho_n, pnl_vect_get(k, n) , pnl_vect_get(theta, k, n))
              + 0.5 * En(T, m, omega, rho_n, pnl_vect_get(k, n) , pnl_vect_get(theta, k, n));
        for (l = 0; l < N; l++)
        {
            for (i = 0; i < M; i++)
                pnl_vect_set(rho_m, i, pnl_mat_get(rho, i, l));
            res += 0.25 * F_n_m(T, m, omega, rho_n, rho_m, pnl_vect_get(k, n), pnl_vect_get(l, k, n))
                  + 0.5 * Dn(T, m, omega, rho_n, pnl_vect_get(k, n), pnl_vect_get(theta, k, n));
        }

    }
    pnl_vect_free(&rho_n);
    pnl_vect_free(&rho_m);
    return res;
}

static double mu4(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *theta)
{
    PnlVect *rho_n, *rho_m;
    int i, n, l, M, N;
    double res = 0.0;

    N = theta->size;
    M = T->size - 1;
    rho_n = pnl_vect_create(M);
    rho_m = pnl_vect_create(M);

    for (n = 0; n < N; n++)
    {
        for (i = 0; i < M; i++)
            pnl_vect_set(rho_n, i, pnl_mat_get(rho, i, n));

        res += 0.25 * En(T, m, omega, rho_n, pnl_vect_get(k, n) , pnl_vect_get(theta, k, n));

        for (l = 0; l < N; l++)
        {
            for (i = 0; i < M; i++)
                pnl_vect_set(rho_m, i, pnl_mat_get(rho, i, l));
        }
    }
}

```

```

        res += 0.25 * F_n_m(T, m, omega, rho_n, rho_m, pnl_vect_get(k, n), pnl_vect_get(theta, n));
    }

}

pnl_vect_free(&rho_n);
pnl_vect_free(&rho_m);
return res;
}

static double nu(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *k, PnlVect *theta)
{
    return 2.0 * mu1(T, m, omega, rho, k, theta);
}

/* static double nu1(PnlVect *T,PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *k, PnlVect *theta)
/* { */
/*   PnlVect * rho_n, *rho_m; */
/*   int i,n,M,N; */
/*   double res = 0.0; */

/*   N = theta->size; */
/*   M = T->size-1; */
/*   rho_n = pnl_vect_create(M); */
/*   rho_m = pnl_vect_create(M); */

/*   for(n=0;n<N;n++) */
/*       { */
/*           for(i=0;i<M;i++) */
/*               pnl_vect_set(rho_n,i,pnl_mat_get(rho,i,n)); */

/*           res += 0.25*An( T,m, omega,rho_n,pnl_vect_get(k,n) ,pnl_vect_get(theta,n)); */

/*       } */
/*   pnl_vect_free(&rho_n); */
/*   pnl_vect_free(&rho_m); */
/*   return res; */

/* } */

```

```

static double nu2(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *k, PnlVect *theta)
{
    return (mu2(T, m, omega, rho, k, theta) - mu1(T, m, omega, rho, k, theta));
}

static double nu3(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *k, PnlVect *theta)
{
    return mu3(T, m, omega, rho, k, theta);
}

static double nu4(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *k, PnlVect *theta)
{
    PnlVect *rho_n, *rho_m;
    int i, n, M, N;
    double res = 0.0, add = 0.0;

    N = theta->size;
    M = T->size - 1;
    rho_n = pnl_vect_create(M);
    rho_m = pnl_vect_create(M);

    res += mu4(T, m, omega, rho, k, theta);

    for (n = 0; n < N; n++)
    {
        for (i = 0; i < M; i++)
            pnl_vect_set(rho_n, i, pnl_mat_get(rho, i, n));

        add += Cn(T, m, omega, rho_n, pnl_vect_get(k, n), pnl_vect_get(theta, n));
    }

    res += add * add / 8.0;

    pnl_vect_free(&rho_n);
    pnl_vect_free(&rho_m);

    return res;
}

```



```

}

static double nu5(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *
{
    PnlVect *rho_n, *rho_m;
    int i, n, M, N;
    double res = 0.0, add = 0.0;

    N = theta->size;
    M = T->size - 1;
    rho_n = pnl_vect_create(M);
    rho_m = pnl_vect_create(M);

    for (n = 0; n < N; n++)
    {
        for (i = 0; i < M; i++)
            pnl_vect_set(rho_n, i, pnl_mat_get(rho, i, n));

        add += Cn(T, m, omega, rho_n, pnl_vect_get(k, n) , pnl_vect_get(theta, n)

    }
    res += add * add / 4.0;

    pnl_vect_free(&rho_n);
    pnl_vect_free(&rho_m);
    return res;
}

static double nu6(PnlVect *T, PnlVect *m, PnlVect *omega, PnlMat *rho, PnlVect *
{
    PnlVect *rho_n, *rho_m;
    int i, n, M, N;
    double res = 0.0, add = 0.0;

    N = theta->size;
    M = T->size - 1;
    rho_n = pnl_vect_create(M);

```

```

rho_m = pnl_vect_create(M);

for (n = 0; n < N; n++)
{
    for (i = 0; i < M; i++)
        pnl_vect_set(rho_n, i, pnl_mat_get(rho, i, n));

    add += Cn(T, m, omega, rho_n, pnl_vect_get(k, n) , pnl_vect_get(theta, n)

}

res += add * add / 8.0;

pnl_vect_free(&rho_n);
pnl_vect_free(&rho_m);
return res;
}

static void RowFromFile(char *chaine, int numCol, PnlVect *res)
{
    int i = 0;
    char delims[] = "\ t";
    char *result = NULL;
    result = strtok(chaine, delims);
    while (result != NULL)
    {
        pnl_vect_set(res, i, atof(result));

        result = strtok(NULL, delims);
        i++;
    }
}

static PnlMat *ReadFilMatrix(FILE *FVParams)
{
    PnlVect *aux;

```

```

char chaine[1000] = "";
int NumberMat = 1, i, j;
int TAILLE_MAX = 1000;

if (FVParams != NULL)
{
    if (fgets(chaine, TAILLE_MAX, FVParams) != NULL)
        NumberMat = (int) atof(chaine);

    ResBergomi = pnl_mat_create(NumberMat + 1, 3);

    aux = pnl_vect_create(3);
    pnl_mat_set(ResBergomi, 0, 0, (double) NumberMat);

    for (j = 1; j < NumberMat + 1; j++)
    {
        if (fgets(chaine, TAILLE_MAX, FVParams) != NULL)
        {
            RowFromFile(chaine, NumberMat, aux);

            for (i = 0; i < 3; i++)
            {
                pnl_mat_set(ResBergomi, j, i, pnl_vect_get(aux, i));
            }
        }
    }

    fclose(FVParams);
    pnl_vect_free(&aux);
}

return ResBergomi;
}

static int getIndex(PnlMat *FVParams, double T)
{
    int i = 1;

```

```

    if (pnl_mat_get(FVParams, 1, 0) > T)
        return 1;

    while (i <= (int)pnl_mat_get(FVParams, 0, 0) && pnl_mat_get(FVParams, i, 0) <=
        i++);

    return i - 1;
}

static double CallPrice(double S, double K, PnlVect *T, PnlVect *m, PnlVect *ome
{
    double z0, z1, z2, z3, z4;
    double d1, d2, v, st, CB;
    double nu_2, nu_3, nu_4, nu_5, nu_6;

    v = nu(T, m, omega, rho, k , theta);
    nu_2 = nu2(T, m, omega, rho, k , theta);
    nu_3 = nu3(T, m, omega, rho, k , theta);
    nu_4 = nu4(T, m, omega, rho, k , theta);
    nu_5 = nu5(T, m, omega, rho, k , theta);
    nu_6 = nu6(T, m, omega, rho, k , theta);

    z4 = nu_6;
    z3 = z4 - nu_5;
    z2 = z3 + nu_4;
    z1 = z2 - nu_3;
    z0 = z1 + nu_2;

    if (K == 0.0)
        return S;
    else
    {
        d1 = (log(S / K) + v / 2) / sqrt(v);
        d2 = (log(S / K) - v / 2) / sqrt(v);
        st = z0 * HermitePoly(0, -d2) + z1 * HermitePoly(1, -d2) / pow(sqrt(v), 1
        CB = S * cdf_nor(d1) - K * cdf_nor(d2);
        //-----
        return CB + S * cdf_nor(d1) * (nu_2 + nu_3 + nu_4 + nu_5 + nu_6) + K * pnl
    }
}

```

```

}

//-----the Greek: Delta dCallprice/dS-----
static double DeltaCall(double S, double K, PnlVect *T, PnlVect *m, PnlVect *ome
{
    double h;
    h = 0.000001;

    return (CallPrice(S * (1.0 + h), K, T, m, omega, rho, k, theta) - CallPric
}

static void CallPutBergomiRev(FILE *fvParams, double k1, double k2, double Thet
{

    PnlMat *ForVar;
    int Index, i, M;
    PnlMat *rho;
    PnlVect *omega, *m, *T, *theta, *k;
    ForVar = ReadFilMatrix(fvParams);

    Index = getIndex(ForVar, t);
    M = Index;
    if (pnl_mat_get(ForVar, Index, 0) < t)
        M ++;

    rho = pnl_mat_create(M, 2);
    theta = pnl_vect_create(2);
    k = pnl_vect_create(2);
    omega = pnl_vect_create(M);
    m = pnl_vect_create(M);
    T = pnl_vect_create(M + 1);

    pnl_vect_set(theta, 0, 1.0 - Theta);
    pnl_vect_set(theta, 1, Theta);
    pnl_vect_set(k, 0, k1);
    pnl_vect_set(k, 1, k2);

    for (i = 0; i < M; i++)
    {
        pnl_mat_set(rho, i, 0, RhoSX);
    }
}

```

```

    pnl_mat_set(rho, i, 1, RhoSY);
    if (i < M - 1)
        pnl_vect_set(T, i + 1, pnl_mat_get(ForVar, i + 1, 0));
        pnl_vect_set(omega, i, pnl_mat_get(ForVar, i, 1));
        pnl_vect_set(m, i, pnl_mat_get(ForVar, i, 2));
    }
    pnl_vect_set(m, 0, pnl_mat_get(ForVar, 1, 2));
    pnl_mat_set(rho, M - 1, 0, RhoSX);
    pnl_mat_set(rho, M - 1, 1, RhoSY);

    pnl_vect_set(T, 0, 0.0000);
    pnl_vect_set(omega, 0, 0.0000);

    pnl_vect_set(T, M, t);

    *price = exp(-r * t) * CallPrice(S * exp((r - q) * t), K, T, m, omega, rho, k);
    *delta = exp(-q * t) * DeltaCall(S * exp((r - q) * t), K, T, m, omega, rho, k);

    pnl_mat_free(&rho);
    pnl_vect_free(&theta);
    pnl_vect_free(&k);
    pnl_vect_free(&omega);
    pnl_vect_free(&m);
    pnl_vect_free(&T);
    pnl_mat_free(&ResBergomi);
}

//----- return a pointer of Call price and
//----- a pointer of Delta
//-----

int ApExpansionBergomiReviOA(double S0, NumFunc_1 *p, double t, double r, double q)
{
    double K, price, delta;
    int flag_call;
    FILE *FVPARAMS = NULL;
    FVPARAMS = fopen(ForwardVarianceData, "r");

    if ((p->Compute) == &Call)

```

[illegible]

```

        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE));
    }

static int CHK_OPT(AP_EXPANSION_BERGOMIREV)(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->HelpFilenameHint = "AP_EXPANSION_BERGOMIREV";
        Met->init = 1;
    }

    return OK;
}

PricingMethod MET(AP_EXPANSION_BERGOMIREV) =
{
    "AP_EXPANSION_BERGOMIREV",
    {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_EXPANSION_BERGOMIREV),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID} ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_EXPANSION_BERGOMIREV),
    CHK_ok,
    MET(Init)
};

```