

[Help](#)

```

#include "bs2d_std2d.h"
#include "pnl/pnl_cdf.h"
#define PRECISION 1.0e-7 /*Precision for the localization of FD methods*/

int PutMinAn(double s1, double s2, double k, double t,
             double r, double divid1, double divid2,
             double sigma1, double sigma2, double rho,
             double *ptprice, double *ptdelta1, double *ptdelta2)
{
    double b1, b2, sigma, rho1, rho2, d, d1, d2, c0, c1;

    b1 = r - divid1;
    b2 = r - divid2;
    sigma = sqrt(SQR(sigma1) + SQR(sigma2) - 2 * rho * sigma1 * sigma2);
    if (((sigma - PRECISION) <= 0.) && ((rho + PRECISION) >= 1.))
    {
        if ((s1 * exp(-divid1 * t)) <= (s2 * exp(-divid2 * t)))
        {
            pnl_cf_put_bs(s1, k, t, r, divid1, sigma1, ptprice, ptdelta1);
            *ptdelta2 = 0.;
        }
        else
        {
            pnl_cf_put_bs(s2, k, t, r, divid2, sigma2, ptprice, ptdelta2);
            *ptdelta1 = 0.;
        }
    }
    else
    {
        rho1 = (sigma1 - rho * sigma2) / sigma;
        rho2 = (sigma2 - rho * sigma1) / sigma;
        d = (log(s1 / s2) + (b1 - b2 + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
        d1 = (log(s1 / k) + (b1 + SQR(sigma1) / 2.0) * t) / (sigma1 * sqrt(t));
        d2 = (log(s2 / k) + (b2 + SQR(sigma2) / 2.0) * t) / (sigma2 * sqrt(t));

        c0 = s1 * exp((b1 - r) * t) * (1.0 - cdf_nor(d)) + s2 * exp((b2 - r) * t)
        c1 = s1 * exp((b1 - r) * t) * pnl_cdf2nor(d1, -d, -rho1)
            + s2 * exp((b2 - r) * t) * pnl_cdf2nor(d2, d - sigma * sqrt(t), -rho2)
            - k * exp(-r * t) * pnl_cdf2nor(d1 - sigma1 * sqrt(t), d2 - sigma2 *

```

```

    /*Price*/
    *ptprice = k * exp(-r * t) - c0 + c1;

    /*Deltas*/
    *ptdelta1 = exp((b1 - r) * t) * (cdf_nor(d) - 1.0) + exp((b1 - r) * t) * p
    *ptdelta2 = -exp((b2 - r) * t) * cdf_nor(d - sigma * sqrt(t)) + exp((b2 -
    }
    return 0;
}

int CALC(CF_PutMin)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid1, divid2;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid1 = log(1. + ptMod->Divid1.Val.V_DOUBLE / 100.);
    divid2 = log(1. + ptMod->Divid2.Val.V_DOUBLE / 100.);

    return PutMinAn(ptMod->S01.Val.V_PDOUBLE, ptMod->S02.Val.V_PDOUBLE, (ptOpt->Pa
        ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
        r, divid1, divid2,
        ptMod->Sigma1.Val.V_PDOUBLE, ptMod->Sigma2.Val.V_PDOUBLE, ptMo
        &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE), &(Me
    }

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

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

```

```
    return OK;
}

PricingMethod MET(CF_PutMin) =
{
    "CF_PutMin",
    {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(CF_PutMin),
    { {"Price", DOUBLE, {100}, FORBID}, {"Delta1", DOUBLE, {100}, FORBID} , {"Delta2", DOUBLE, {100}, FORBID} },
    {{" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(CF_PutMin),
    CHK_ok,
    MET(Init)
} ;
```