

[Help](#)

```
#include <stdlib.h>
#include "
href../../mod/bs1d/bs1d_std/bs1d_std_h_src.pdfbs1d_std.h"
#include "
href../../common/error_msg_h_src.pdferror_msg.h"

static int CoxRossRubinstein_79(int am, double s, NumFunc_1 *p, double t, doubl
{
    int i, j;
    double u, d, h, pu, pd, a1, stock, upperstock;
    double *P, *iv;

    /*Price, intrinsic value arrays*/
    P = malloc((N + 1) * sizeof(double));
    if (P == NULL)
        return MEMORY_ALLOCATION_FAILURE;
    iv = malloc((2 * N + 1) * sizeof(double));
    if (iv == NULL)
        return MEMORY_ALLOCATION_FAILURE;

    /*Up and Down factors*/
    h = t / (double)N;
    a1 = exp(h * (r - divid));
    u = exp(sigma * sqrt(h));
    d = 1. / u;

    /*Risk-Neutral Probability*/
    pu = (a1 - d) / (u - d);
    pd = 1. - pu;

    if ((pd >= 1.) || (pd <= 0.))
        return NEGATIVE_PROBABILITY;

    pu *= exp(-r * h);
    pd *= exp(-r * h);
    /*Intrinsic value initialisation*/
    upperstock = s;
    for (i = 0; i < N; i++)
```

```

    upperstock *= u;
    stock = upperstock;
    for (i = 0; i < 2 * N + 1; i++)
    {
        iv[i] = (p->Compute)(p->Par, stock);
        stock *= d;
    }

    /*Terminal Values*/
    for (j = 0; j <= N; j++)
        P[j] = iv[2 * j];

    /*Backward Resolution*/
    for (i = 1; i <= N - 1; i++)
        for (j = 0; j <= N - i; j++)
        {
            P[j] = pu * P[j] + pd * P[j + 1];
            if (am)
                P[j] = MAX(iv[i + 2 * j], P[j]);
        }

    /*Delta*/
    *ptdelta = (P[0] - P[1]) / (s * u - s * d);

    /*First time step*/
    P[0] = pu * P[0] + pd * P[1];
    if (am)
        P[0] = MAX(iv[N], P[0]);

    /*Price*/
    *ptprice = P[0];

    free(P);
    free(iv);

    return OK;
}

int CALC(TR_CoxRossRubinstein)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;

```

```

    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid;

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

    return CoxRossRubinstein_79(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
                                ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Maturity.Val
                                r, divid, ptMod->Sigma.Val.V_PDOUBLE, Met->Par[0].
}

static int CHK_OPT(TR_CoxRossRubinstein)(void *Opt, void *Mod)
{
    return OK;
}

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

        Met->Par[0].Val.V_INT2 = 100;
    }

    return OK;
}

PricingMethod MET(TR_CoxRossRubinstein) =
{
    "TR_CoxRossRubinstein",
    {"StepNumber", INT2, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(TR_CoxRossRubinstein),
    {"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
    CHK_OPT(TR_CoxRossRubinstein),
    CHK_tree,
    MET(Init)
};

```