

[Help](#)

```
#include <stdlib.h>
#include "bs1d_std.h"
#include "error_msg.h"

static int ExtendedCRR(int am, double s, NumFunc_1 *p, double t, double r, double sigma)
{
    int i, j, extN = N + 2;
    double u, d, h, pu, pd, a1, stock, upperstock;
    double *P, *iv;

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

    /*Up and Down factors*/
    h = t / (double)N; /*N and not extN: as if one starts 2 periods before*/
    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 initialization*/
    upperstock = s;
    for (i = 0; i < extN; i++) /*As if one starts 2 periods before*/
        upperstock *= u;
```

```

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

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

/*Backward Resolution*/
for (i = 1; i <= extN - 2; i++) /*Not extN-1 since we stop the tree at the sec
    for (j = 0; j <= extN - 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[2]) / (s * u * u - s * d * d); /*Points above and below*/

/*Price*/
*ptprice = P[1]; /*Middle point*/

/*Desallocation*/
free(P);
free(iv);

return OK;
}

int CALC(TR_ExtendedCRR)(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 ExtendedCRR(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
                       ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Maturity.Val.V_DATE -
                       r, divid, ptMod->Sigma.Val.V_PDOUBLE, Met->Par[0].Val.V_INT2);
}

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

```