

## [Help](#)

```
#include <stdlib.h>
#include "
href../../mod/bs1d/bs1d_limdisc/bs1d_limdisc_h_src.pdfbs1d_limdisc.h"

static double *pu, *pd, *pm, *u, *d, *m;
static double *alpha, *eta, *e;

static int calibration(int N, double s, double down_barrier, double dlog_s, double
{
    double M, V;
    int i;

    alpha[0] = s;
    for (i = 1; i <= N; i++)
    {
        /*Time Dependent Central Nodes*/
        e[i] = floor((log(down_barrier) + 0.5 * dlog_s - log(alpha[i - 1]))) / dlog_s;
        alpha[i] = exp(log(down_barrier) + 0.5 * dlog_s - e[i] * dlog_s);
        eta[i - 1] = alpha[i] / alpha[i - 1];

        /*Up,Down,Middle Factor*/
        u[i - 1] = up_factor * eta[i - 1];
        d[i - 1] = down_factor * eta[i - 1];
        m[i - 1] = eta[i - 1];

        /*Probabilities*/
        M = exp((r - divid) * dt) / eta[i - 1];
        V = exp(2.*(r - divid) * dt) * (exp(sigma2 * dt) - 1.) / SQR(eta[i - 1]);
        pu[i - 1] = (up_factor * (V + SQR(M) - M) - (M - 1.)) / ((up_factor - 1.));
        pd[i - 1] = (SQR(up_factor) * (V + SQR(M) - M) - CUB(up_factor) * (M - 1.));
        pm[i - 1] = 1. - pu[i - 1] - pd[i - 1];
    }
    return OK;
}

static int tr_cheuckvorst(int am, double t, NumFunc_1 *p, double r, double divid)
{
    double sigma2, K;
```

```

int i, j;
double lambda, dt, dlog_s, rebate;
double up_factor, down_factor;
double price, downer_stock, stock1;
int npoints;
double *P, *Old_P, *iv, *stock, *vect_t, *monit_date;
int monit_flag;
int N;

N = nb_mon_date * M;
K = p->Par[0].Val.V_PDOUBLE;

/*Price, intrinsic value arrays*/
P = malloc((2 * N + 2) * sizeof(double));
Old_P = malloc((2 * N + 2) * sizeof(double));
iv = malloc((2 * N + 2) * sizeof(double));
u = malloc((2 * N + 2) * sizeof(double));
d = malloc((2 * N + 2) * sizeof(double));
m = malloc((2 * N + 2) * sizeof(double));
pu = malloc((2 * N + 2) * sizeof(double));
pd = malloc((2 * N + 2) * sizeof(double));
pm = malloc((2 * N + 2) * sizeof(double));
stock = malloc((2 * N + 2) * sizeof(double));
alpha = malloc((2 * N + 2) * sizeof(double));
e = malloc((2 * N + 2) * sizeof(double));
eta = malloc((2 * N + 2) * sizeof(double));
vect_t = malloc((2 * N + 2) * sizeof(double));
monit_date = malloc((nb_mon_date + 1) * sizeof(double));

lambda = 0.5 * sqrt(2.*M_PI);
sigma2 = SQR(sigma);
rebate = 0.;
dt = t / (double)N;

/*Monitoring Dates*/
for (i = 1; i <= nb_mon_date; i++)
    monit_date[i] = ((double)i) * (t) / (double)nb_mon_date;

/*Calibration*/
dlog_s = lambda * sigma * sqrt(dt);
up_factor = exp(dlog_s);

```

```

down_factor = 1. / up_factor;
calibration(N, s, down_barrier, dlog_s, up_factor, down_factor, r, divid, dt,

for (i = 1; i <= N; i++)
    vect_t[i] = ((double)i) * (t) / (double)N;

stock1 = s;
for (i = 0; i < -e[1]; i++)
    stock1 *= d[i];

/*Maturity Condition*/
stock1 = s;
for (i = 0; i < N; i++)
    stock1 *= d[i];

downer_stock = stock1;
for (i = 0; i <= 2 * N; i++)
{
    stock[i] = stock1;
    if ((stock1 <= down_barrier))
        P[i] = rebate;
    else
        P[i] = MAX(0., stock1 - K);
    Old_P[i] = P[i];
    iv[i] = MAX(0., stock1 - K);
    stock1 *= u[N - 1];
}

/*Backward Induction*/
npoints = 2 * N + 1;
for (i = N - 1; i > 0; i--)
{
    downer_stock *= u[i];
    monit_flag = 0;

    for (j = 1; j < nb_mon_date; j++)
        if (fabs((vect_t[i]) - monit_date[j]) < 0.00000000001)
            monit_flag = 1;

    npoints -= 2;
    stock1 = downer_stock;

```

```

for (j = 0; j < npoints; j++)
{

    if (monit_flag == 0)
    {
        P[j] = exp(-r * dt) * (pd[i] * Old_P[j] + pm[i] * Old_P[j + 1] + p
        if (am)
            P[j] = MAX(iv[j + i], Old_P[j]);
    }
    else if (monit_flag == 1)
    {
        if ((stock1 <= down_barrier))
        {
            P[j] = rebate;
        }
        else
        {
            P[j] = exp(-r * dt) * (pd[i] * Old_P[j] + pm[i] * Old_P[j + 1]
            if (am)
                P[j] = MAX(iv[j + i], P[j]);
        }
    }
    stock1 *= u[N - 1];
}
for (j = 0; j < npoints; j++)
    Old_P[j] = P[j];

if (i == 1)
    /*Delta*/
    *pt_delta = (P[2] - P[0]) / (s * up_factor - s * down_factor);
}

/*Last Step*/
price = exp(-r * dt) * (pd[0] * P[0] + pm[0] * P[1] + pu[0] * P[2]);
if (am)
    price = MAX(s - K, price);

/*Price*/
*pt_price = price;

```



```

}

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

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

        Met->Par[0].Val.V_INT2 = 25;
    }

    return OK;
}

PricingMethod MET(TR_CK) =
{
    "TR_CheuckVorst",
    {"TimeStepNumber for Monitoring Period", INT2, {100}, ALLOW}, {" ", PREMIA_NU
    CALC(TR_CK),
    {"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
    CHK_OPT(TR_CK),
    CHK_ok,
    MET(Init)
} ;

```