

[Help](#)

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

static int MMS(int am, double s, NumFunc_1 *p, double tt, double rr, double divi
{
    int i, j, k;
    double u, inv_u, d, h, pu, pd, stock, upperstock;
    double a, b, c;
    double alpha, beta, gamma1, K0;
    double *P;

    /*Price, intrinsic value arrays*/
    P = malloc((N + 1) * sizeof(double));

    /*Strike*/
    K0 = p->Par[0].Val.V_PDOUBLE;

    /*Compute index k*/
    h = tt / (double)N;
    k = N / 2;

    /*Compute pu*/
    alpha = (rr - divid - 0.5 * SQR(sigma)) * h;
    beta = log(K0 / s);
    gamma1 = SQR(sigma) * h;
    a = -SQR(beta) + 2.*k * beta * alpha + 2.*(N - k) * beta * alpha - gamma1 * SQ
    b = SQR(beta) + SQR(k * alpha) - 2.*k * beta * alpha - SQR((N - k) * alpha) -
    c = SQR((N - k) * alpha) - gamma1 * SQR(k) - gamma1 * SQR(N) + 2 * k * N * gam

    pu = (-b - sqrt(SQR(b) - 4.*a * c)) / (2.*a);

    pd = 1 - pu;

    /*Compute up,down factor*/
    d = exp((pu * beta - alpha * (N - k)) / (N * (pu - 1) + k));
    u = exp((beta * ((N - k) * (pu - 1)) + alpha * (N - k) * k) / ((N - k) * (N *
```

```

inv_u = 1. / u;

pu = pu * exp(-rr * h);
pd = pd * exp(-rr * h);

/*Intrinsic value initialisation*/
upperstock = s;
for (i = 0; i < N; i++)
    upperstock = upperstock * u;

/*Terminal Values*/
stock = upperstock;
for (i = 0; i < N + 1; i++)
{
    P[i] = (p->Compute)(p->Par, stock);
    stock = stock * inv_u * d;
}

/*Backward Resolution*/
for (i = 1; i <= N - 1; i++)
{
    upperstock = upperstock * inv_u;
    stock = upperstock;
    for (j = 0; j <= N - i; j++)
    {
        P[j] = pu * P[j] + pd * P[j + 1];
        if (am)
            P[j] = MAX((p->Compute)(p->Par, stock), P[j]);
        stock = stock * inv_u * d;
    }
}

/*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((p->Compute)(p->Par, s), P[0]);

```

```

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

    free(P);
    return OK;
}

static int MMSR(int am, double s, NumFunc_1 *p, double t, double r, double divid)
{
    double price1, delta1, price2, delta2;

    MMS(am, s, p, t, r, divid, sigma, N, &price1, &delta1);
    MMS(am, s, p, t, r, divid, sigma, N / 2, &price2, &delta2);

    /*Price*/
    *ptprice = 2.*price1 - price2;

    /*Delta*/
    *ptdelta = 2.*delta1 - delta2;

    return OK;
}

int CALC(TR_MMSR)(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 MMSR(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE, ptOpt->PayOff.V
}

static int CHK_OPT(TR_MMSR)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PutAmer") == 0) || (strcmp(((Option *)Opt)
        return OK;

    return WRONG;
}

```

```

}

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

```