

[Help](#)

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

static int BBS(double s, NumFunc_1 *p, double t, double r, double divid, double
{
    int i, j;
    double u, d, h, pu, pd, a1, b1, stock, upperstock, delta, tmp, d_square;
    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));
    b1 = SQR(a1) * (exp(SQR(sigma) * h) - 1.);
    tmp = SQR(a1) + b1 + 1.;
    u = (tmp + sqrt(SQR(tmp) - 4.*SQR(a1))) / (2.*a1);
    d = 1. / u;
    d_square = d * d;

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

/*Backward Resolution*/
stock = upperstock * d;

/*LastTime Step*/
for (j = 0; j < N; j++)
{
    if ((p->Compute) == &Call)
        pnl_cf_call_bs(stock, p->Par[0].Val.V_PDOUBLE, h, r, divid, sigma, P + j)
    else if ((p->Compute) == &Put)
        pnl_cf_put_bs(stock, p->Par[0].Val.V_PDOUBLE, h, r, divid, sigma, P + j,
        stock *= d_square;
    P[j] = MAX(iv[1 + 2 * j], P[j]);
}

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

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

free(P);
free(iv);

```

```

    return OK;
}

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

    BBS(s, p, t, r, divid, sigma, N, &price1, &delta1);
    BBS(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_BBSR)(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 BBSR(ptMod->S0.Val.V_PDDOUBLE, ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Mat
}

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