

[Help](#)

```

/* We need Nd1 here */
#define USE_ND1 1
#include "bs1d_std.h"
#define INC 1.0e-5 /*Relative Increment for Delta-Hedging*/

/*Put McMillan Exponent*/
static double McMillanPut_Exp(double r, double divid, double sigma, double T)
{
    double ratio = 2.0 * (r - divid) / (sigma * sigma);
    double delta = (ratio - 1.0);

    delta = SQR(delta) + 8.0 * (1.0 + r * T) / (sigma * sigma * T);

    return 0.5 * (1. - ratio - sqrt(delta));
}

/*Call McMillan Exponent*/
static double McMillanCall_Exp(double r, double divid, double sigma, double T)
{
    double ratio = 2.0 * (r - divid) / (sigma * sigma);
    double delta = (ratio - 1.0);

    delta = SQR(delta) + 8.0 * (1.0 + r * T) / (sigma * sigma * T);

    return 0.5 * (1. - ratio + sqrt(delta));
}

/*Put Critical Price*/
static double Contact_PointPut(double r, double divid, double sigma, double T, d
                                double (*exponent_method)(double, double, double,
{
    const double precision = 0.00001;
    double previous;
    double exponent = (*exponent_method)(r, divid, sigma, T);
    double current = K;
    double put_price, put_delta;

    do
    {

```

```

        previous = current;
        pnl_cf_put_bs(previous, K, T, r, divid, sigma, &put_price, &put_delta);
        current = -exponent * (K - put_price) / ((1. - exp(-divid * T) * Nd1(previous, T, r, sigma)));
    }
    while (!(fabs((previous - current) / current) <= precision));

    return current;
}

/*Call Critical Price*/
static double Contact_PointCall(double r, double divid, double sigma, double T,
                                double (*exponent_method)(double, double, double))
{
    const double precision = 0.00001;
    double previous;
    double exponent = (*exponent_method)(r, divid, sigma, T);
    double current = K;
    double call_price, call_delta;

    do
    {
        previous = current;
        pnl_cf_call_bs(previous, K, T, r, divid, sigma, &call_price, &call_delta);
        current = exponent * (K + call_price) / (-(1. - exp(-divid * T) * Nd1(previous, T, r, sigma)));
    }
    while (!(fabs((previous - current) / current) <= precision));

    return current;
}

/*McMillan Formula*/
static double Formula_McMillan(double r, double divid, double sigma, double T, double K)
{
    double exponent;
    double critical_price;
    double a, put_price, put_delta, call_price, call_delta;

    if ((p->Compute) == &Put)
    {
        critical_price = Contact_PointPut(r, divid, sigma, T, K, McMillanPut_Exp);
    }
    else
    {
        critical_price = Contact_PointCall(r, divid, sigma, T, K, McMillanCall_Exp);
    }
}

```

```

    if (x < critical_price)
    {
        return (K - x);
    }
    else
    {
        exponent = McMillanPut_Exp(r, divid, sigma, T);
        a = critical_price * (1. - exp(-divid * T) * Nd1(critical_price, r, divid, sigma, T));
        pnl_cf_put_bs(x, K, T, r, divid, sigma, &put_price, &put_delta);
        return put_price + a * pow(x / critical_price, exponent);
    }
}

else if ((p->Compute) == &Call)
{
    critical_price = Contact_PointCall(r, divid, sigma, T, K, McMillanCall_Exp(r, divid, sigma, T));
    if (x >= critical_price)
    {
        return (x - K);
    }
    else
    {
        exponent = McMillanCall_Exp(r, divid, sigma, T);
        a = critical_price * (1. - exp(-divid * T) * Nd1(critical_price, r, divid, sigma, T));
        pnl_cf_call_bs(x, K, T, r, divid, sigma, &call_price, &call_delta);
        return call_price + a * pow(x / critical_price, exponent);
    }
}

/*Never reached normally*/
return 0.;
}

static int McMillan_81(double s, NumFunc_1 *p, double t, double r, double divid,
{
    double s_plus, s_minus;

    s_plus = s * (1. + INC);
    s_minus = s * (1. - INC);

    /*Price*/
    *ptprice = Formula_McMillan(r, divid, sigma, t, s, p->Par[0].Val.V_DOUBLE, p);
}

```

```

    /*Delta*/
    *ptdelta = (Formula_McMillan(r, divid, sigma, t, s_plus, p->Par[0].Val.V_DOUBL

    return OK;
}

int CALC(AP_McMillan)(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 McMillan_81(ptMod->S0.Val.V_PDOUBLE,
                       ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Maturity.Val.V_DATE -
                       r, divid, ptMod->Sigma.Val.V_PDOUBLE,
                       &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE));
}

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

    return WRONG;
}

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

    return OK;
}

```

```
PricingMethod MET(AP_McMillan) =  
{  
    "AP_McMillan",  
    {{" ", PREMIA_NULLTYPE, {0}, FORBID}},  
    CALC(AP_McMillan),  
    {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR  
    CHK_OPT(AP_McMillan),  
    CHK_ok ,  
    MET(Init)  
} ;
```