

[Help](#)

```
#include "bs2d_std2d.h"

static int ProductTR(int am,double s1,double s2,
    NumFunc_2* p,double T, double r,double divid1,double
    divid2,double sigma1,double sigma2,double rho,int
    N,double *ptprice,double *ptdelta1,double *ptdelt
    a2)
{
    double stock1,stock2,lowerstock1,lowerstock2,
    u1,u2,d1,d2,scan1,scan2,puu,pud;
    double h;
    double iv;
    int i,j,k,dummy;
    double **P;

    /*2D Price Array allocation*/
    P=(double **)calloc(N+1,sizeof(double*));
    if (P==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    for (i=0;i<N+1;i++)
    {
        P[i]=(double *)calloc(N+1,sizeof(double))
    ;
        if (P[i]==NULL)
            return MEMORY_ALLOCATION_FAILURE;
    }

    /*Up and Down factors*/;
    h=T/(double)N;
    u1=exp(((r-divid1)-sigma1*sigma1/2.)*h);
    u2=exp(((r-divid2)-sigma2*sigma2/2.)*h);
    scan1=exp(sigma1*sqrt(h));
    scan2=exp(sigma2*sqrt(h));
    d1=u1/scan1;
    u1=u1*scan1;
    d2=u2/scan2;
    u2=u2*scan2;
    scan1=scan1*scan1;
    scan2=scan2*scan2;
```

```

/*Risk-Neutral probabilities*/
puu=exp(-r*h)*(1.+rho)/4.;
pud=exp(-r*h)*(1.-rho)/4.;

lowerstock1=s1;
lowerstock2=s2;
for (i=0;i<N;i++)
{
    lowerstock1*=d1;
    lowerstock2*=d2;
}

/*Terminal prices*/
stock1=lowerstock1;stock2=lowerstock2;
for (i=0;i<N+1;i++,stock1*=scan1,stock2=low
erstock2)
    for (j=0;j<N+1;j++,stock2*=scan2)
        P[i][j]=(p->Compute)(p->Par,stock1,
stock2);

/*Backward scheme*/
for (k=N;k>=2;k--)
{
    lowerstock1/=d1;lowerstock2/=d2;
    stock1=lowerstock1;
    for (i=0;i<k;i++,stock1*=scan1,stock2
=lowerstock2)
        for (j=0;j<k;j++,stock2*=scan2)
        {
            P[i][j]=puu*(P[i][j]+P[i+1][
j+1])+pud*(P[i+1][j]+P[i][j+1]);
            if (am)
            {
                iv=(p->Compute)(p->Par,
stock1,stock2);
                P[i][j]=MAX(iv,P[i][j]);
            }
        }
    }

/*Deltas*/

```

```

        dummy=Delta_Operator(u1,d1,u2,d2,s1,s2,P[
1][1],P[1][0],P[0][1],P[0][0],ptdelta1,ptdelta2)
;

        /*First Time Step*/
        P[0][0]=puu*(P[0][0]+P[1][1])+pud*(P[0][1
]+P[1][0]);
        if (am)
        {
            iv=(p->Compute)(p->Par,s1,s2);
            P[0][0]=MAX(iv,P[0][0]);
        }
        /*Price*/
        *ptprice=P[0][0];

        /*2D Price Array deallocation*/
        for (i=0;i<N+1;i++)
            free(P[i]);
        free(P);

        return OK;
    }

int CALC(TR_ProductTR)(void *Opt,void *Mod,PricingMethod *Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    double r,divid1,divid2;

    r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
    divid1=log(1.+ptMod->Divid1.Val.V_DOUBLE/100.
);
    divid2=log(1.+ptMod->Divid2.Val.V_DOUBLE/100.
);

    return ProductTR(ptOpt->EuOrAm.Val.V_BOOL,pt
Mod->S01.Val.V_PDOUBLE,
        ptMod->S02.Val.V_PDOUBLE,ptOpt->PayOff.
Val.V_NUMFUNC_2,
        ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.

```

```

V_DATE,r,divid1,divid2,
    ptMod->Sigma1.Val.V_PDDOUBLE,ptMod->Sigma2
    .Val.V_PDDOUBLE,ptMod->Rho.Val.V_RGDOUBLE,Met->
    Par[0].Val.V_INT,&(Met->Res[0].Val.V_DOUBLE),&(Met->
    Res[1].Val.V_DOUBLE),&(Met->Res[2].Val.V_
    DOUBLE) );
}

int CHK_OPT(TR_ProductTR)(void *Opt, void *Mod)
{
    Option* ptOpt=(Option*)Opt;
    TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

    return OK;
}

static int MET(Init)(PricingMethod *Met)
{
    static int first=1;

    if (first)
    {
        Met->Par[0].Val.V_INT2=100;

        first=0;
    }

    return OK;
}

PricingMethod MET(TR_ProductTR)=
{
    "TR_ProductTR",
    {"StepNumber",INT2,100,ALLOW},{" ",END,0
    ,FORBID}},
    CALC(TR_ProductTR),
    {"Price",DOUBLE,100,FORBID},{"Delta1",
    DOUBLE,100,FORBID} ,{"Delta2",DOUBLE,100,FORBID} ,
    {" ",END,0,FORBID}},
    CHK_OPT(TR_ProductTR),
    CHK_tree,

```

```
    MET(Init)
};
```

References