

[Help](#)

```
#include "bs2d_std2d.h"

static int Euler(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)
{
    int i,j,k;
    double sigma11,sigma12,sigma21,sigma22,h;
    double a11,a12,a21,a22,m1,m2,u1,u2,a,c,d,iv,x
    1,x2,proba,trend1,trend2,scan1,scan2;
    double *temp1,**temp2,**P;

    /*Memory Allocation*/
    temp1=(double *)calloc(2*N+1,sizeof(double));
    if (temp1==NULL)
        return MEMORY_ALLOCATION_FAILURE;

    temp2=(double **)calloc(2*N+1,sizeof(double *
    ));
    if (temp2==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    for (i=0;i<2*N+1;i++)
    {
        temp2[i]=(double *)calloc(2*N+1,sizeof(
        double));
        if (temp2[i]==NULL)
            return MEMORY_ALLOCATION_FAILURE;
    }

    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;
    }
}
```

```

}

/*Variance-Covariance Matrix*/
sigma11=sigma1;
sigma12=0.0;
sigma21=rho*sigma2;
sigma22=sigma2*sqrt(1.0-SQR(rho));
u1=r-divid1;
u2=r-divid2;
a11=SQR(sigma11)+SQR(sigma12);
a12=sigma11*sigma21+sigma12*sigma22;
a21=a12;
a22=SQR(sigma21)+SQR(sigma22);
m1=u1-a11/2.0;
m2=u2-a22/2.0;

/*Up and Down factors*/
h=t/(double)N;
a=sigma11*sqrt(h);
c=sigma21*sqrt(h);
d=sigma22*sqrt(h);
x1=log(s1);
x2=log(s2);

/*Probability*/
proba=exp(-r*h)/4.0;

/*Terminal Values*/
trend1=exp(x1+m1*t);
trend2=exp(x2+m2*t);
for (j=0;j<=2*N;j++)
    temp1[j]=exp((double)(-N+j)*a);
for (i=0;i<=2*N;i++)
    for (j=0;j<=2*N;j++)
        temp2[i][j]=exp((double)(-N+j)*c+d*(
double)(N-i));

    for (i=0;i<=N;i++)
        for (j=0;j<=N;j++)
            P[i][j]=(p->Compute) (p->Par,trend1*temp1[2*j],trend2*temp2[2*i][2*j]);

```

```

/*Backward Cycle*/
scan1=exp(-m1*h);
scan2=exp(-m2*h);

for (k=1;k<=N-1;k++)
{
    trend1*=scan1;
    trend2*=scan2;
    for(i=0;i<=N-k;i++)
    {
        for(j=0;j<=N-k;j++)
        {
            P[i][j]=proba*(P[i][j]+P[
i][j+1]+P[i+1][j]+P[i+1][j+1]);
            if (am)
            {
                iv=(p->Compute)(p->
Par,trend1*temp1[2*j+k],trend2*temp2[2*i+k][2*j+k])
;
                P[i][j]= MAX(iv,P[i][
j]);
            }
        }
    }
}

/*Deltas*/
*ptdelta1=0.;
*ptdelta2=0.;

/*First Time Step*/
trend1*=scan1;
trend2*=scan2;
P[0][0]=proba*(P[0][0]+P[0][1]+P[1][0]+
P[1][1]);
            if (am)
            {
                iv=(p->Compute)(p->
Par,trend1*temp1[N],trend2*temp2[N][N]);

```

```

                                P[0][0]= MAX(iv,P[0][
0]);
                                }

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

    /*Memory desallocation*/
    free(temp1);

    for (i=0;i<2*N+1;i++)
        free(temp2[i]);
    free(temp2);

    for (i=0;i<N+1;i++)
        free(P[i]);
    free(P);

    return OK;
}

int CALC(TR\_Euler)(void *Opt,void *Mod,Pricing
    Method *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/10
0.);

    return Euler(ptOpt->EuOrAm.Val.V_BOOL,ptMod-
>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_PDOU

```

```

        BLE,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_Euler)(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_Euler)=
{
    "TR_Euler",
    {"StepNumber",INT2,100,ALLOW},{ " ",END,0,FO
RBID}},
    CALC(TR_Euler),
    {"Price",DOUBLE,100,FORBID},{"Delta1",
DOUBLE,100,FORBID} ,{"Delta2",DOUBLE,100,FORBID} ,
    {" ",END,0,FORBID}},
    CHK_OPT(TR_Euler),
    CHK_tree,
    MET(Init)
}

```

};

References