

[Help](#)

```
#include "bs2d_std2d.h"

static int Explicit(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 *ptde
    lta2)
{
    int M,TimeIndex,j,i,Index;
    double x1,x2,sigma11,sigma12,sigma21,sigma22,
    m1,m2,p1,p2;
    double k,h,limit,trend1,trend2,scan1,scan2,iv
    ;
    double **P,**G,*temp1,**temp2;

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

    temp2=(double **)calloc(N+1,sizeof(double *))
    ;
    if (temp2==NULL)
```

```

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

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

    /*Covariance Matrix*/
    sigma11=sigma1;
    sigma12=0.0;
    sigma21=rho*sigma2;
    sigma22=sigma2*sqrt(1.0-SQR(rho));

    m1=(r-divid1)-SQR(sigma11)/2.0;
    m2=(r-divid2)-(SQR(sigma21)+SQR(sigma22))/2.0
    ;

    /*Space Localisation*/
    limit=sqrt(t)*sqrt(log(1/PRECISION));

    /*Space Step*/
    h=2.*limit/(double) N;

    /*Stability Condition Time Step*/
    k=SQR(h)/(2.-r*SQR(h));
    M=(int) (t/k);

    /*Probabilities*/
    p1=1.-k*(2./SQR(h)+r);
    p2=k/(2.0*SQR(h));

    /*Terminal Values*/
    x1=log(s1);
    x2=log(s2);
    trend1=exp(x1+m1*t);
    trend2=exp(x2+m2*t);

```

```

for (i=0;i<=N;i++)
    temp1[i]=exp(sigma11*(-limit+h*i));
for(i=1;i<N;i++) {
    for (j=1;j<N;j++) {
        temp2[i][j]=exp(sigma21*(-limit+h*(
double)j)+sigma22*(limit-h*(double)i));
        P[i][j]= (p->Compute)(p->Par, trend1*
temp1[j],trend2*temp2[i][j]);
    }
}

```

```

for(i=0;i<=N;i++)
{
    G[i][0]=0.;
    G[i][N]=0.;
    G[0][i]=0.;
    G[N][i]=0.;
}

```

```

/*Finite Difference Cycle */
scan1=exp(-m1*k);
scan2=exp(-m2*k);
for (TimeIndex=1;TimeIndex<=M;TimeIndex++) {
    trend1*=scan1;
    trend2*=scan2;
    for(i=1;i<N;i++)
        for(j=1;j<N;j++) G[i][j]=P[i][j];

    for(i=1;i<N;i++)
        for(j=1;j<N;j++)
        {
            P[i][j]=p1*G[i][j]+p2*(G[i+1]
[j]+G[i][j+1]+G[i-1][j]+G[i][j-1]);
            /*Splitting for the american
case */
            if (am)
            {
                iv=(p->Compute) (p->Par,
trend1*temp1[j],trend2*temp2[i][j]);
                P[i][j]=MAX(iv,P[i][j]);
            }
        }
    }
}

```

```

    }
}

Index=(int)((double)N/2.0);

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

/*Deltas*/
*ptdelta2=(P[Index-1][Index]-P[Index+1][Index])/
(2.*s2*h*sigma22);
*ptdelta1=((P[Index][Index+1]-P[Index][Index-1])/
(2.*s1*h)-sigma21*(s2/s1)*( *ptdelta2))/sigma11;

/*Memory desallocation*/
for (i=0;i<N+1;i++)
    free(P[i]);
free(P);

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

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

free(temp1);

return OK;
}

int CALC(FD_Explicit)(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/100.
    );

    return Explicit(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_PDOU
    BLE,ptMod->Sigma2.Val.V_PDOUBLE,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(FD_Explicit)(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(FD_Explicit)=
{
  "FD_Explicit",
  {{ "StepNumber",INT2,100,ALLOW},{ " ",END,0
,FORBID}},
  CALC(FD_Explicit),
  {{ "Price",DOUBLE,100,FORBID},{ "Delta1",
DOUBLE,100,FORBID} ,
  { "Delta2",DOUBLE,100,FORBID} ,
  { " ",END,0,FORBID}},
  CHK_OPT(FD_Explicit),
  CHK_ok,
  MET(Init)
};
```

## References