

[Help](#)

```
#include "bs1d_std.h"

static int BrennanSchwartz_79(double s, NumFunc_1
    *p, double t, double r, double divid, double sig
    ma, int N, int M, double *ptprice, double *ptdelta)
{
    int Index, PriceIndex, TimeIndex;
    double k, vv, l, h, z, alpha, beta, gamma, y, temp, upw
        ind_alphacoef;
    double *Obst, *A, *B, *C, *P;

    /*Memory Allocation*/
    if (N%2==1) N++;
    Obst=(double *)malloc(N*sizeof(double));
    if (Obst==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    A=(double *)malloc(N*sizeof(double));
    if (A==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    B=(double *)malloc(N*sizeof(double));
    if (B==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    C=(double *)malloc(N*sizeof(double));
    if (C==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    P=(double *)malloc(N*sizeof(double));
    if (P==NULL)
        return MEMORY_ALLOCATION_FAILURE;

    /*Time Step*/
    k=t/(double)M;

    /*Space Localisation*/
    vv=0.5*sigma*sigma;
    z=(r-divid)-vv;
    l=sigma*sqrt(t)*sqrt(log(1.0/PRECISION))+fabs(z
        *t);

    /*Space Step*/
    h=2.0*l/(double)N;
```

```

/*Peclet Condition-Coefficient of diffusion augmented */
if ((h*fabs(z))<=vv)
    upwind_alphacoef=0.5;
else
{
    if (z>0.) upwind_alphacoef=0.0;
    else if (z<=0.) upwind_alphacoef=1.0;
}
vv-=z*h*(upwind_alphacoef-0.5);

/*Lhs Factor of implicate-schema*/
alpha=k*(-vv/(h*h)+z/(2.0*h));
beta=1.0+k*(r+2.*vv/(h*h));
gamma=k*(-vv/(h*h)-z/(2.0*h));

if ( (p->Compute) == &Call)
{
    temp=alpha;
    alpha=gamma;
    gamma=temp;
}

for(PriceIndex=0;PriceIndex<=N-1;PriceIndex++)
{
    A[PriceIndex]=alpha;
    B[PriceIndex]=beta;
    C[PriceIndex]=gamma;
}

/*Neumann Boundary Condition*/
B[0]=beta+alpha;
B[N-1]=beta+gamma;

/*Gauss Algorithm*/
for(PriceIndex=N-2;PriceIndex>=0;PriceIndex--)
    B[PriceIndex]=B[PriceIndex]-C[PriceIndex]*A[PriceIndex+1]/B[PriceIndex+1];
for(PriceIndex=0;PriceIndex<N;PriceIndex++)
    A[PriceIndex]=A[PriceIndex]/B[PriceIndex];

```

```

for(PriceIndex=0;PriceIndex<N;PriceIndex++)
    C[PriceIndex]=C[PriceIndex]/B[PriceIndex+1];

/*Terminal Values*/
y=log(s);
for(PriceIndex=0;PriceIndex<N;PriceIndex++)
{
    if ( (p->Compute) == &Put)
        Obst[PriceIndex]=(p->Compute)(p->Par,exp(
y-1+(double)(PriceIndex+1)*h));
    else
        Obst[PriceIndex]=(p->Compute)(p->Par,exp(
y+1-(double)(PriceIndex+1)*h));
    P[PriceIndex]= Obst[PriceIndex];
}

/*Finite Difference Cycle*/
for(TimeIndex=1;TimeIndex<=M;TimeIndex++)
{
    /*First Loop*/
    for(PriceIndex=N-2;PriceIndex>=0;PriceIndex--)
        P[PriceIndex]=P[PriceIndex]-C[PriceIndex]
        *P[PriceIndex+1];

    /*Second Loop*/
    P[0]/= B[0];
    for(PriceIndex=1;PriceIndex<N;PriceIndex++)
    {
        P[PriceIndex]=P[PriceIndex]/B[PriceIndex]-
        A[PriceIndex]*P[PriceIndex-1];

        P[PriceIndex]=MAX(Obst[PriceIndex],P[PriceIndex]);
    }
}
Index=(int) floor ((double)(N-1)/2.0);

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

```

```

/*Delta */
if ( (p->Compute) == &Put)
*ptdelta=(P[Index+1]-P[Index-1])/(2.0*s*h);
else
*ptdelta=(P[Index-1]-P[Index+1])/(2.0*s*h);

/*Memory Desallocation*/
free(Obst);
free(A);
free(B);
free(C);
free(P);

return OK;
}

int CALC(FD_BrennanSchwartz)(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 BrennanSchwartz_79(ptMod->S0.Val.V_PDOUN
BLE,ptOpt->PayOff.Val.V_NUMFUNC_1,
ptOpt->Maturity.Val.
V_DATE-ptMod->T.Val.V_DATE,r,divid,ptMod->Sigma.
Val.V_PDOUNBLE,
Met->Par[0].Val.V_
INT,Met->Par[1].Val.V_INT,
&(Met->Res[0].Val.V_
DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
}

int CHK_OPT(FD_BrennanSchwartz)(void *Opt, void *
Mod)
{

```

```

    if ( (strcmp( ((Option*)Opt)->Name,"CallAmer")=
        =0) || (strcmp( ((Option*)Opt)->Name,"PutAmer")=
        =0) )
        return OK;

    return WRONG;
}

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

    if (first)
    {
        Met->Par[0].Val.V_INT2=100;
        Met->Par[1].Val.V_INT2=100;

        first=0;
    }

    return OK;
}

PricingMethod MET(FD_BrennanSchwartz)=
{
    "FD_BrennanSchwartz",
    {"SpaceStepNumber",INT2,100,ALLOW    },{"TimeStepNumber",INT2,100,ALLOW},{" " ,END,0,FORBID}},
    CALC(FD_BrennanSchwartz),
    {"Price",DOUBLE,100,FORBID},{"Delta",DOUBLE,100,FORBID} ,{" " ,END,0,FORBID}},
    CHK_OPT(FD_BrennanSchwartz),
    CHK_fdiff,
    MET(Init)
};

```

References