```
   Help
#include <stdlib.h>
#include  "bs1d_std.h"
#include "error_msg.h"
#define PRECISION 1.0e-7 /*Precision for the localization
    of FD methods*/

static int Explicit(int am,double s,NumFunc_1  *p,double t,
    double r,double divid,double sigma,int N, double *ptprice,
    double *ptdelta)
{
  int i,j,M,Index;
  double   h,z,k,p1,p2,p3,l,x,vv,upwind_alphacoef;
  double *S,*P,*Obst;

  /*Memory Allocaction*/
  if (N%2==1) N++ ;
  S= malloc((N+1)*sizeof(double));
  if (S==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  P= malloc((N+1)*sizeof(double));
  if (P==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  Obst= malloc((N+1)*sizeof(double));
  if (Obst==NULL)
    return MEMORY_ALLOCATION_FAILURE;

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

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

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

```
}
vv-=z*h*(upwind_alphacoef-0.5);

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

/*"Probabilities" associated to points*/
p1=k*(vv/(SQR(h))-z/(2.0*h));
p2=1.0-k*(2.*vv/SQR(h)+r);
p3=k*(vv/(SQR(h))+z/(2.0*h));

/*Terminal Values*/
x=log(s);
for (i=0;i<=N;i++)
  {
    P[i]=(p->Compute)(p->Par,exp(x-l+(double)i*h));
    Obst[i]=P[i];
  }

/*Finite Difference Cycle*/
for(i=1;i<=M;i++)
  {
    for (j=1;j<N;j++)
      {
        S[j]=p1*P[j-1]+p2*P[j]+p3*P[j+1];
  /*Splitting for American case*/
        if (am)
          S[j]= MAX(Obst[j],S[j]);
      }
    for (j=1;j<N;j++)
      P[j]=S[j];
  }

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

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

/*Delta*/
*ptdelta  = (P[Index+1]-P[Index-1])/(2.0*s*h);
```

```
  /*Memory Desallocation*/
  free(S);
  free(P);
  free(Obst);

  return OK;
}

int CALC(FD_Explicit)(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 Explicit(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S0.Val.V_
    PDOUBLE,
                  ptOpt->PayOff.Val.V_NUMFUNC_1,ptOpt->Matu
    rity.Val.V_DATE-ptMod->T.Val.V_DATE,
                  r,divid,ptMod->Sigma.Val.V_PDOUBLE,Met->
    Par[0].Val.V_INT,&(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].
    Val.V_DOUBLE));
}

static int CHK_OPT(FD_Explicit)(void *Opt, void *Mod)
{
  return  OK;
}

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

      Met->Par[0].Val.V_INT2=100;
```

```
    }

  return OK;
}

PricingMethod MET(FD_Explicit)=
{
  "FD_Explicit",
  {{"TimeStepNumber",INT2,{100},ALLOW},{" ",PREMIA_NULLTYP
    E,{0},FORBID}},
  CALC(FD_Explicit),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
    ID} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_Explicit),
  CHK_tree,
  MET(Init)
};
```

References