```
   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 howard_amer1(double s,NumFunc_1  *p,double t,
    double r,double divid, double sigma,
      int N,int M,double theta,double epsilon,double *pt
    price,double *ptdelta)
{
  double k,z,vv,l,h,x,alpha,beta,gamma,alpha1,beta1,gamma1,
    upwind_alphacoef,temp,error,g0,g1;
  int i,j,Index;
  double *P,*Obst,*R,*A,*B,*C,*G;
  int *pp;

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

```
  return MEMORY_ALLOCATION_FAILURE;
pp= malloc((N+1)*sizeof(int));
if (pp==NULL)
  return MEMORY_ALLOCATION_FAILURE;

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

/*Space Localisation*/
z=(r-divid)-SQR(sigma)/2.0;
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 */
vv=0.5*SQR(sigma);
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);

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

alpha1=k*(1.0-theta)*(vv/(h*h)-z/(2.0*h));
beta1=1.0-k*(1.0-theta)*(r+2.*vv/(h*h));
gamma1=k*(1.0-theta)*(vv/(h*h)+z/(2.0*h));

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

```
/*Finite Difference Cycle*/
for (i=1;i<=M;i++)
  {
    /*Init Control*/
    for (j=0;j<=N;j++)
pp[j]=0;

    for(j=1;j<N;j++)
R[j]=P[j]*beta1+alpha1*P[j-1]+gamma1*P[j+1];

    /*Howard Cycle*/
    do
{
  error=0.;

  for (j=1;j<N;j++)
    {
      g0=P[j-1]*alpha+P[j]*beta+P[j+1]*gamma-R[j];
      g1=P[j]-Obst[j];
      if (g0<g1) pp[j]=0;else pp[j]=1;
    }

  for (j=1;j<N;j++)
    {
      if (pp[j]==0)
  {
    G[j]=R[j];A[j]=alpha;B[j]=beta;C[j]=gamma;
  }
      else {
  G[j]=Obst[j];A[j]=0;B[j]=1;C[j]=0;
      }
    }

  /*Set Gauss*/
  for(j=N-2;j>=1;j--)
    B[j]=B[j]-C[j]*A[j+1]/B[j+1];
  for(j=1;j<N;j++)
    A[j]=A[j]/B[j];
  for(j=1;j<N-1;j++)
    C[j]=C[j]/B[j+1];
```

```
    /*Solve the system*/
    for(j=N-2;j>=1;j--)
      G[j]=G[j]-C[j]*G[j+1];

    temp=P[1];
    P[1] =G[1]/B[1];
    error=fabs(P[1]-temp);
    for(j=2;j<N;j++)
      {
        temp=P[j];
        P[j]=G[j]/B[j]-A[j]*P[j-1];
        error+=fabs(P[j]-temp);
      }
  }
      while (error>epsilon);
      /*End Howard Cycle*/
    }
  /*End Finite Difference Cycle*/

  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(P);
  free(pp);
  free(G);
  free(A);
  free(B);
  free(C);
  free(R);
  free(Obst);

  return OK;

}
```

```
int CALC(FD_Howard_amer1)(void *Opt,void *Mod,Pricing
    Method *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 howard_amer1(ptMod->S0.Val.V_PDOUBLE,ptOpt->PayO
    ff.Val.V_NUMFUNC_1,
          ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DATE,
    r,divid,ptMod->Sigma.Val.V_PDOUBLE,
          Met->Par[0].Val.V_INT,Met->Par[1].Val.V_INT,
    Met->Par[2].Val.V_RGDOUBLE,
          Met->Par[3].Val.V_RGDOUBLE,
          &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.
    V_DOUBLE));
}


static int CHK_OPT(FD_Howard_amer1)(void *Opt, void *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);


  if ((opt->EuOrAm). Val.V_BOOL==AMER)
    return OK;

  return WRONG;
}


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

```
    {
      Met->init=1;


      Met->Par[0].Val.V_INT2=100;
      Met->Par[1].Val.V_INT2=100;
      Met->Par[2].Val.V_RGDOUBLE=0.5;
      Met->Par[3].Val.V_RGDOUBLE=0.000001;


    }

  return OK;
}


PricingMethod MET(FD_Howard_amer1)=
{
  "FD_Howard",
  {{"SpaceStepNumber",INT2,{128},ALLOW},
   {"TimeStepNumber",INT2,{128},ALLOW},
   {"Theta",RGDOUBLE,{100},ALLOW},
   {"Epsilon",RGDOUBLE,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_Howard_amer1),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_Howard_amer1),
  CHK_fdiff,
  MET(Init)
};
```

References