```
   Help
#include  "bs2d_std2d.h"
#include "error_msg.h"
#define PRECISION 1.0e-7 /*Precision for the localization
    of FD methods*/

static void init(double **g,double **u,double alpha,double
    beta,double gam,int flag,int N)
{
  int i,j;

  if (flag==1)
    {
      for(i=1;i<N;i++)
  for(j=1;j<N;j++)
    g[i][j]=alpha*u[i][j-1]+beta*u[i][j]+gam*u[i][j+1];
    }
  else
    {
      for(i=1;i<N;i++)
  for(j=1;j<N;j++)
    g[j][i]=alpha*u[j-1][i]+beta*u[j][i]+gam*u[j+1][i];
    }

  return;
}

static void swap(double **u,double **g,int N)
{
  int i,j;

  for(i=1;i<N;i++)
    for(j=1;j<N;j++)
      u[i][j]=g[i][j];

  return;
}

static int tri(double **g,double alpha,double beta,double
    gam,int flag,int N)
{
```

```
int i,j,z;
double *c,*a,*b;

/*Memory Allocation*/
a=(double *)calloc(N+1,sizeof(double));
if (a==NULL)
  return MEMORY_ALLOCATION_FAILURE;
b=(double *)calloc(N+1,sizeof(double));
if (b==NULL)
  return MEMORY_ALLOCATION_FAILURE;
c=(double *)calloc(N+1,sizeof(double));
if (c==NULL)
  return MEMORY_ALLOCATION_FAILURE;

/*Gauss Algorithm*/
b[N-1]=beta;
for(i=N-2;i>=1;i--) b[i]=beta-gam*alpha/b[i+1];
for(i=1;i<=N-1;i++) a[i]=alpha/b[i];
for(i=1;i<N-1;i++) c[i]=gam/b[i+1];
if (flag==1) {
  for(j=1;j<N;j++)
    {
for(z=N-2;z>=1;z--)
  g[z][j]=g[z][j]-g[z+1][j]*c[z];

g[1][j]=g[1][j]/b[1];

for (z=2;z<N;z++)
  g[z][j]=(g[z][j]/b[z]-g[z-1][j]*a[z]);
    }
} else {
  for(j=1;j<N;j++)
    {
for(z=N-2;z>=1;z--)
  g[j][z]=g[j][z]-g[j][z+1]*c[z];

g[j][1]=g[j][1]/b[1];

for (z=2;z<N;z++)
  g[j][z]=(g[j][z]/b[z]-g[j][z-1]*a[z]);
    }
```

```
  }
  /*Memory Desallocation*/
  free(a);
  free(b);
  free(c);

  return OK;
}

static int Adi(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,int M, double *ptprice,double *pt
    delta1,double *ptdelta2)
{
  int i,j,TimeIndex,Index,flag;
  double x1,x2,sigma11,sigma21,sigma22,k,limit,h,m1,m2,tre
    nd1,trend2;
  double a1,b1,a2,b2,scan1,scan2,iv;
  double **P,**G,**temp2,*temp1;

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

/*Time Step*/
k=t/(2.*(double)M);

/*Rhs Factor of first step*/
b1=1.-k*(1./SQR(h)+r/2.0);
a1=k/(2.0*SQR(h));

/*Lhs Factor of second step*/
b2=1.+k*(1./SQR(h)+r/2.0);
a2=-k/(2.0*SQR(h));

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

```
trend2=exp(x2+m2*t);
for (j=0;j<=N;j++)
  temp1[j]=exp(sigma11*(-limit+h*(double)j));
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]);
}
  }

/*Homegenous Dirichlet Conditions*/
for(i=0;i<=N;i++)
  {
    P[i][0]=0.;
    P[i][N]=0.;
    P[0][i]=0.;
    P[N][i]=0.;
  }

/*Finite Difference Cycle */
scan1=exp(-m1*2.*k);
scan2=exp(-m2*2.*k);
for (TimeIndex=1;TimeIndex<=M;TimeIndex++)
  {
    trend1*=scan1;
    trend2*=scan2;

    /*First Step*/
    flag=1;

    /*Init Rhs*/
    init(G,P,a1,b1,a1,flag,N);

    /*Gauss Algorith*/
    tri(G,a2,b2,a2,flag,N);
    swap(P,G,N);

    /*Second Step*/
```

```
    flag=2;

    /*Init Rhs*/
    init(G,P,a1,b1,a1,flag,N);

    /*Gauss Algorithm*/
    tri(G,a2,b2,a2,flag,N);
    swap(P,G,N);

    /*Splitting for the american case */
    if (am)
{
  for(i=1;i<N;i++)
    {
      for(j=1;j<N;j++)
  {
    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_Adi)(void *Opt,void *Mod,PricingMethod *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 Adi(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S01.Val.V_PDO
    UBLE,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_PDOUBLE,ptMod->Sigma2.Val.V_PDO
    UBLE,ptMod->Rho.Val.V_RGDOUBLE,
       Met->Par[0].Val.V_INT,Met->Par[1].Val.V_INT,
       &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.V_
    DOUBLE),&(Met->Res[2].Val.V_DOUBLE) );
}

static int CHK_OPT(FD_Adi)(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;
      Met->Par[1].Val.V_INT2=100;


    }

  return OK;
}

PricingMethod MET(FD_Adi)=
{
  "FD_Adi",
  {{"SpaceStep",INT2,{100},ALLOW},{"TimeStep",INT2,{100},
    ALLOW} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_Adi),
  {{"Price",DOUBLE,{100},FORBID},{"Delta1",DOUBLE,{100},FO
    RBID} ,{"Delta2",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_Adi),
  CHK_ok,
  MET(Init)
};
```

References