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


static int restriction2(int l,double **d,double **u,double
    **f,double aa,double bb)
{
  int nl,i,j;
  double **w;

  nl=pow(2, l+1)-1;

  w=(double **)calloc(nl+2,sizeof(double *));
  if (w==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  for (i=0;i<nl+2;i++)
    {
      w[i]=(double *)calloc(nl+2,sizeof(double));
      if (w[i]==NULL)
  return MEMORY_ALLOCATION_FAILURE;
    }

  for (i=1;i<nl+1;i++)
    for (j=1;j<nl+1;j++)
      w[i][j]=aa*u[i][j]+bb*(u[i+1][j]+u[i-1][j]+u[i][j+1]+
    u[i][j-1])-f[i][j];

  for (i=2;i<nl;i=i+2)
    for (j=2;j<nl;j=j+2)
      d[i/2][j/2]=(((w[i-1][j-1]+w[i+1][j-1]+w[i-1][j+1]+w[
    i+1][j+1])/2.0+w[i][j-1]+w[i+1][j]+w[i-1][j]+w[i][j+1])/2.0
    +w[i][j])/4.0;

  for (i=0;i<nl+2;i++)
    free(w[i]);
  free(w);

  return OK;
```

```
}


static int prolon2(int l,double **u,double **v)
{
  int nl,nl1,i,j;
  double **w;

  nl=pow(2, l+1)-1;
  nl1=pow(2, l)-1;

  w=(double **)calloc(nl+2,sizeof(double *));
  if (w==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  for (i=0;i<nl+2;i++)
    {
      w[i]=(double *)calloc(nl+2,sizeof(double));
      if (w[i]==NULL)
  return MEMORY_ALLOCATION_FAILURE;
    }

  for (i=1;i<nl+1;i=i+2)
    {w[i][0]=w[0][i]=w[nl+1][i]=w[i][nl+1]=0.0;}

  for (i=0;i<nl1+2;i++)
    for (j=0;j<nl1+2;j++)
      w[2*i][2*j]=v[i][j];

  for (i=1;i<nl+1;i=i+2)
    for (j=2;j<nl;j=j+2)
      w[i][j]=(w[i-1][j]+w[i+1][j])/2.0;

  for (i=1;i<nl+1;i++)
    for (j=1;j<nl+1;j=j+2)
      w[i][j]=(w[i][j-1]+w[i][j+1])/2.0;

  for (i=1;i<nl+1;i++)
    for (j=1;j<nl+1;j++)
      u[i][j]=u[i][j]-w[i][j];

  for (i=0;i<nl+2;i++)
```

```
    free(w[i]);
  free(w);

  return OK;
}


static int MGM2(int l,double **u,double **f,double t,
    double r,double divid1,double divid2,double sigma1,double sigma2
    ,double rho,int N,int M)
{
  double h,k,limit,aa,bb;
  double **d,**v;
  int nl,nl1,ii,i,j;

  nl=pow(2, l+1)-1;
  nl1=pow(2, l)-1;

  /*Memory Allocation*/
  d=(double **)calloc(nl1+2,sizeof(double *));
  if (d==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  for (i=0;i<nl1+2;i++)
    {
      d[i]=(double *)calloc(nl1+2,sizeof(double));
      if (d[i]==NULL)
  return MEMORY_ALLOCATION_FAILURE;
    }

  v=(double **)calloc(nl1+2,sizeof(double *));
  if (v==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  for (i=0;i<nl1+2;i++)
    {
      v[i]=(double *)calloc(nl1+2,sizeof(double));
      if (v[i]==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    }


  /*Space Localisation*/
```

```
limit=sqrt(t)*sqrt(log(1./PRECISION));
h=2.*limit/(double)(nl+1);

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

/*Factor*/
aa=1.+2.*k/(h*h)+r*k;
bb=-1.*k/2./(h*h);

if (l==0) {u[1][1]=f[1][1]/aa;}
else
 {
  /* 2 iterations of Gauss-Seidel*/
  for (ii=1;ii<3;ii++)
for (i=1;i<=nl;i++)
 for (j=1;j<=nl;j++)
  u[i][j]=((-u[i+1][j]-u[i-1][j]-u[i][j+1]-u[i][j-1])*
 bb+f[i][j])/aa;

  restriction2(l,d,u,f,aa,bb);

  for (i=0;i<=nl1+1;i++)
for (j=0;j<=nl1+1;j++)
 v[i][j]=0;

  MGM2(l-1,v,d,t,r,divid1,divid2,sigma1,sigma2,rho,N,M)
 ;
  prolon2(l,u,v);

  /* 2 iterations of Gauss-Seidel*/
  for (ii=1;ii<3;ii++)
for (i=1;i<=nl;i++)
 for (j=1;j<=nl;j++)
  u[i][j]=((-u[i+1][j]-u[i-1][j]-u[i][j+1]-u[i][j-1])*
 bb+f[i][j])/aa;
 }

for (i=0;i<nl1+2;i++)
 free(v[i]);
free(v);
```

```
  for (i=0;i<nl1+2;i++)
    free(d[i]);
  free(d);
  return OK;
}


static int mult_euro2(double s1,double s2,NumFunc_2 *p,
    double t,double r,double divid1,double divid2,double sigma1,
    double sigma2,double rho,int l,int M, double *ptprice,double *pt
    delta1,double *ptdelta2)
{
  double h,x1,x2,sigma11,sigma21,sigma22,m1,m2,trend1,trend
    2,limit;
  double **P,**w;
  int Index,TimeIndex,i,j,N;

  /*Memory Allocation*/
  N=pow(2, l+1)-1+1;
  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;
    }

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

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

for (i=1;i<=N;i++)
  for (j=1;j<=N;j++)
    P[i][j]=(p->Compute)(p->Par,trend1*exp(sigma11*(-limi
  t+h*(double)j)),trend2*exp(sigma21*(-limit+h*(double)j)+si
  gma22*(limit-h*(double)i)));

/*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*/
for (TimeIndex=1;TimeIndex<=M;TimeIndex++)
  {
    /*Init*/
    for (i=1;i<=N;i++)
for (j=1;j<=N;j++)
```

```
    w[i][j]=P[i][j];

      /*Multi-grid method*/
      MGM2(l,P,w,t,r,divid1,divid2,sigma1,sigma2,rho,N,M);
    }
  /*End Finite Difference Cycle*/

  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*(*ptdelta2))/sigma11;


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

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

  return OK;
}


int CALC(FD_Multigrid)(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 mult_euro2(ptMod->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_PDOUBLE,ptMod->Sigma2.Val.V_
    PDOUBLE,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_Multigrid)(void *Opt, void *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

  if ((opt->EuOrAm). Val.V_BOOL==EURO)
    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=5;
      Met->Par[1].Val.V_INT2=100;


    }

  return OK;
}

PricingMethod MET(FD_Multigrid)=
{
  "FD_Multigrid_Euro",
  {{"Number of Grids",INT2,{100},ALLOW},{"TimeStep",INT2,{1
    00},ALLOW} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(FD_Multigrid),
  {{"Price",DOUBLE,{100},FORBID},{"Delta1",DOUBLE,{100},FO
    RBID} ,{"Delta2",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_Multigrid),
  CHK_fdiff,
  MET(Init)
};
```

References