```
   Help
#include  "bs2d_std2d.h"

/*Compress Diagonal Storage*/
static void cds(int  n, double a,double b,
    double c, double d,double e, double f,double g,
    double i2, double j2, double **band)
{
  int j,nsr;


  nsr=(int)sqrt(n);

  for (j=nsr+2;j<=n;j++)
    if (((j-1)%(nsr))!=0)
      band[1][j]=j2;

  for (j=nsr+1;j<=n;j++)band[2][j]=d;

  for (j=nsr+1;j<=n;j++)
    if (((j)%(nsr))!=0)
      band[3][j]=f;

  for (j=2;j<=n;j++)
    if (((j-1)%(nsr))!=0)
      band[4][j]=c;

  for (j=1;j<=n;j++) band[5][j]=a;

  for (j=1;j<=n;j++)
    if(((j)%(nsr))!=0)
      band[6][j]=b;

  for (j=2;j<=n-nsr;j++)
    if (((j-1)%(nsr))!=0)
      band[7][j]=g;

  for (j=1;j<=n-nsr;j++) band[8][j]=e;

  for (j=1;j<=n-nsr;j++)
    if ((j%(nsr))!=0)
```

```
      band[9][j]=i2;

  return;
}

/*Dirichlet Boundary Conditions*/
static void Dirichlet(int  N, double a2,double b2
    , double c2, double d2,double e2, double f2,
    double g2, double i2, double j2,double x1,double x2,
    double limit1,double limit2,double h1,double h2,
    NumFunc_2  *p,double *bound)
{
  int i,j,N1,Ns;


  N1=N-1;
  Ns=SQR(N1);

  for(i=1;i<Ns;i++) bound[i]=0.;

  bound[1]=j2*(p->Compute)(p->Par, exp(x1-limit1)
    ,exp(x2+limit2))+
    d2*(p->Compute)(p->Par, exp(x1-limit1+h1),exp
    (x2+limit2))+
    f2*(p->Compute)(p->Par, exp(x1-limit1+2.*h1),
        exp(x2+limit2))+c2*(p->Compute)(p->Par,
     exp(x1-limit1),exp(x2+limit2-h2))+
    g2*(p->Compute)(p->Par, exp(x1-limit1+h1),exp
    (x2+limit2-2.*h2));

  for(i=2;i<N-1;i++)
    bound[i]=j2*(p->Compute)(p->Par, exp(x1-limi
    t1+h1*(double)(i-1)),exp(x2+limit2))+
      d2*(p->Compute)(p->Par, exp(x1-limit1+h1*(
    double)i),exp(x2+limit2))+
      f2*(p->Compute)(p->Par, exp(x1-limit1+h1*(
    double)(i+1)),exp(x2+limit2));

  bound[N-1]=j2*(p->Compute)(p->Par, exp(x1-limi
    t1+h1*(double)(N-2)),exp(x2+limit2))+
    d2*(p->Compute)(p->Par, exp(x1-limit1+h1*(
```

```
  double)(N-1)),exp(x2+limit2))+
  f2*(p->Compute)(p->Par, exp(x1-limit1+h1*(
  double)N),exp(x2+limit2))+
  b2*(p->Compute)(p->Par, exp(x1-limit1+h1*(
  double)N),exp(x2+limit2-h2))+
  i2*(p->Compute)(p->Par, exp(x1-limit1+h1*(
  double)N),exp(x2+limit2-2.*h2));

N1=N-1;
j=1;
for(i=N;i<Ns-N1;i=i+N1)  {
  bound[i]=j2*(p->Compute)(p->Par,exp(x1-limit1
  ),exp(x2+limit2-h2*(double)(j-1)))+
    c2*(p->Compute)(p->Par,exp(x1-limit1),exp(x
  2+limit2-(double)(j)))+
    g2*(p->Compute)(p->Par,exp(x1-limit1),exp(x
  2+limit2-(double)(j+1)));
  j++;
}

j=1;
for(i=2*N1;i<Ns-N1;i=i+N1) {
  bound[i]=f2*(p->Compute)(p->Par, exp(x1+limi
  t1),exp(x2+limit2-h2*(double)(j-1)))+
    b2*(p->Compute)(p->Par,exp(x1+limit1),exp(x
  2+limit2-(double)(j)))+
    i2*(p->Compute)(p->Par,exp(x1+limit1),exp(x
  2+limit2-(double)(j+1)));
  j++;
}

bound[Ns-N1+1]=j2*(p->Compute)(p->Par, exp(x1-
  limit1),exp(x2-limit2+2.*h2))+
  c2*(p->Compute)(p->Par, exp(x1-limit1),exp(x2
  -limit2+h2))+
  g2*(p->Compute)(p->Par, exp(x1-limit1),exp(x2
  -limit2))+
  e2*(p->Compute)(p->Par, exp(x1-limit1+h1),exp
  (x2-limit2))+
  i2*(p->Compute)(p->Par, exp(x1-limit1+2.*h1),
   exp(x2-limit2));
```

```
  for(i=1;i<N-1;i++)
    bound[Ns-N1+1+i]=g2*(p->Compute)(p->Par, exp(
    x1-limit1+h1*(double)i),exp(x2-limit2))+
      e2*(p->Compute)(p->Par, exp(x1-limit1+h1*(
    double)(i+1)),exp(x2-limit2))+
      i2*(p->Compute)(p->Par, exp(x1-limit1+h1*(
    double)(i+2)),exp(x2-limit2));

  bound[Ns]=g2*(p->Compute)(p->Par, exp(x1+limit1
    -h1*(double)2),exp(x2-limit2))+
    e2*(p->Compute)(p->Par, exp(x1+limit1+h1),exp
    (x2-limit2))+
    i2*(p->Compute)(p->Par, exp(x1+limit1),exp(x2
    -limit2))+
    b2*(p->Compute)(p->Par, exp(x1+limit1),exp(x2
    +limit2-h2))+
    f2*(p->Compute)(p->Par, exp(x1+limit1),exp(x2
    +limit2-2.*h2));

  return;
}

static int GMRES(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,int max_iter,double tol,int m,int precon
    d,double *ptprice,double *ptdelta1,double *ptdelt
    a2)
{
  int TimeIndex,j,i,Index,dummy;
  int Ns;
  double x1,x2,m1,m2,cov;
  double limit1,limit2,h1,h2;
  double a2,b2,c2,d2,e2,f2,g2,i2,j2;
  double k;
  double *P,*b,*Obst,*bound,*pivots,**H,**band;
  x2=0.0;
  x1=0.0;
  /*Memory Allocation*/
  Ns=(N-1)*(N-1);
```

```
P=(double *)calloc(Ns+1,sizeof(double));
if (P==NULL)
  return MEMORY_ALLOCATION_FAILURE;


b=(double *)calloc(Ns+1,sizeof(double));
if (b==NULL)
  return MEMORY_ALLOCATION_FAILURE;

Obst=(double *)calloc(Ns+1,sizeof(double));
if (Obst==NULL)
  return MEMORY_ALLOCATION_FAILURE;

bound=(double *)calloc(Ns+1,sizeof(double));
if (bound==NULL)
  return MEMORY_ALLOCATION_FAILURE;

pivots=(double *)calloc(Ns+1,sizeof(double));
if (pivots==NULL)
  return MEMORY_ALLOCATION_FAILURE;

band=(double**)calloc(10,sizeof(double*));
if (band==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<10;i++)
  {
    band[i]=(double *)calloc(Ns+1,sizeof(
  double));
    if (band[i]==NULL)
return MEMORY_ALLOCATION_FAILURE;
  }

m1=(r-divid1)-SQR(sigma1)/2.0;
m2=(r-divid2)-SQR(sigma2)/2.0;
cov=rho*sigma1*sigma2;

/*Space Localisation*/
limit1=sigma1*sqrt(t)*sqrt(log(1/PRECISION))+fa
  bs(m1)*t;
limit2=sigma2*sqrt(t)*sqrt(log(1/PRECISION))+fa
```

```
  bs(m2)*t;

/*Space Step*/
h1=2.*limit1/(double) N;
h2=2.*limit2/(double)N;

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

/*Lhs factor*/
a2=1.+k*(r+SQR(sigma1)/SQR(h1)+SQR(sigma2)/SQR(
  h2));
b2=-k*(SQR(sigma1)/(2.*SQR(h1))+m1/(2.*h1));
c2=-k*(SQR(sigma1)/(2.*SQR(h1))-m1/(2.*h1));
d2=-k*(SQR(sigma2)/(2.*SQR(h2))+m2/(2.*h2));
e2=-k*(SQR(sigma2)/(2.*SQR(h2))-m2/(2.*h2));

f2=-k*cov/(4.*h1*h2);
g2=-k*cov/(4.*h1*h2);
i2=k*cov/(4.*h1*h2);
j2=k*cov/(4.*h1*h2);

/*CDS format*/
cds(Ns,a2,b2,c2,d2,e2,f2,g2,i2,j2,band);

/*Preconditioners*/
if (precond==1)
  Diagonal_Precond(band,Ns,pivots);
else
  ILU_Precond(band,Ns,pivots);

/*Dirichlet Boundary Conditions*/
Dirichlet(N,a2,b2,c2,d2,e2,f2,g2,i2,j2,x1,x2,
  limit1,limit2,h1,h2,
    p,bound);

/*Terminal Values*/
x1=log(s1);
x2=log(s2);

for(i=1;i<N;i++) {
```

```
  for (j=1;j<N;j++) {
    P[(i-1)*(N-1)+j]=(p->Compute)(p->Par, exp(x
  1-limit1+h1*(double)j),
          exp(x2+limit2-h2*(double)i));
    Obst[(i-1)*(N-1)+j]=P[(i-1)*(N-1)+j];
  }
}

/*Finite Difference Cycle */
for (TimeIndex=1;TimeIndex<=M;TimeIndex++)
  {
    /*Rhs Term*/
    for(i=1;i<=Ns;i++)
b[i]=P[i]-bound[i];

    /*Memory Allocation of H*/
    H=(double**)calloc(m+1,sizeof(double*));
    if (H==NULL)
return MEMORY_ALLOCATION_FAILURE;

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

    /*GMRES Algorithm*/
    dummy=gmres(H,band,P,b,m,precond,Ns,max_it
  er,tol,pivots);

    /*Memory desallocation of H*/
    for (i=0;i<m+1;i++)
free(H[i]);
    free(H);

    /*Splitting for American case*/
    if (am)
for(i=1;i<=Ns;i++)
  P[i]=MAX(P[i],Obst[i]);
  }
```

```
  Index=(int)((double)(N-1)/2.0);
  Index=Index*(N-1)+(Index+1);

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


  /*Deltas*/
  *ptdelta1=(P[Index+1]-P[Index-1])/(2.*s1*h1);
  *ptdelta2=(P[Index-(N-1)]-P[Index+(N-1)])/(2.*
    s2*h2);

  /*Memory desallocation*/
  free(P);
  free(b);
  free(Obst);
  free(bound);
  free(pivots);
  for (i=0;i<10;i++)
    free(band[i]);
  free(band);

  return OK;
}

int CALC(FD_GMRES)(void *Opt,void *Mod,Pricing
    Method *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 GMRES(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S0
    1.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_PD
     OUBLE,ptMod->Sigma2.Val.V_PDOUBLE,ptMod->Rho.Val.
     V_RGDOUBLE,
          Met->Par[0].Val.V_INT,Met->Par[1].Val.
     V_INT,Met->Par[2].Val.V_INT,Met->Par[3].Val.V_PD
     OUBLE,Met->Par[4].Val.V_INT,Met->Par[5].Val.V_
     INT,
          &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[
     1].Val.V_DOUBLE),&(Met->Res[2].Val.V_DOUBLE) );
}

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

  return  OK;
}

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;
      Met->Par[2].Val.V_INT2=50;
      Met->Par[3].Val.V_PDOUBLE=0.000001;
      Met->Par[4].Val.V_INT=2;
      Met->Par[5].Val.V_INT=1;
      first=0;
    }

  return OK;
}

PricingMethod MET(FD_GMRES)=
{
```

```
  "FD_GMRES",
  {{"SpaceStepNumber",INT2,100,ALLOW},{"TimeStep
    Number",INT2,100,ALLOW}
   ,{"Max Iter",INT2,100,ALLOW},{"Tol",PDOUBLE,10
    0,ALLOW},{"Restart Number",INT,100,ALLOW},{"Prec
    ondtioner:(1)Diagonal (2)ILU",INT,100,ALLOW},{" "
    ,END,0,FORBID}},
  CALC(FD_GMRES),
  {{"Price",DOUBLE,100,FORBID},{"Delta1",DOUBLE,1
    00,FORBID} ,
   {"Delta2",DOUBLE,100,FORBID} ,
   {" ",END,0,FORBID}},
  CHK_OPT(FD_GMRES),
  CHK_ok,
  MET(Init)
};
```

References