```
   Help
#include  "bs2d_std2d.h"
#include "error_msg.h"

static int BoyleEvnineGibbs_88(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,
                                double *ptprice,double *ptde
    lta1,double *ptdelta2)
{
  int i,j,k;
  double h,m1,m2,u1,u2,d1,d2,lowerstock1,upperstock2,puu,
    pud,pdu,pdd,stock1,stock2;
  double **iv,**P;

  /*Memory Allocation*/
  iv=(double **)calloc(2*N+1,sizeof(double *));
  if (iv==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  for (i=0;i<2*N+1;i++)
    {
      iv[i]=(double *)calloc(2*N+1,sizeof(double));
      if (iv[i]==NULL)
  return MEMORY_ALLOCATION_FAILURE;
    }

  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;
    }

  /*Up and Down factors*/
  h=t/(double)N;
  u1=exp(sigma1*sqrt(h));
  d1= 1.0/u1;
  u2=exp(sigma2*sqrt(h));
```

```
d2= 1.0/u2;

/*Risk-Neutral Probabilities*/
m1=(r-divid1)-SQR(sigma1)/2.0;
m2=(r-divid2)-SQR(sigma2)/2.0;

puu=exp(-r*h)*(1+rho+sqrt(h)*(m1/sigma1+m2/sigma2))/4.0;
pud=exp(-r*h)*(1-rho+sqrt(h)*(m1/sigma1-m2/sigma2))/4.0;
pdu=exp(-r*h)*(1-rho+sqrt(h)*(-m1/sigma1+m2/sigma2))/4.0;
pdd=exp(-r*h)*(1+rho-sqrt(h)*(m1/sigma1+m2/sigma2))/4.0;

/*Terminal Values*/
lowerstock1=s1;upperstock2=s2;
for(i=0;i<N;i++)
  {
    lowerstock1*=d1;
    upperstock2*=u2;
  }

stock1=lowerstock1;stock2=upperstock2;
for (i=0;i<=2*N;i++,stock2*=d2,stock1=lowerstock1)
  for (j=0;j<=2*N;j++,stock1*=u1)
    iv[i][j]=(p->Compute)(p->Par,stock1,stock2);
for (i=0;i<=N;i++)
  for (j=0;j<=N;j++)
    P[i][j]=iv[2*i][2*j];

/*Backward Cycle*/
for (k=1;k<=N-1;k++)
  {
    for(i=0;i<=N-k;i++)
{
  for(j=0;j<=N-k;j++)
    {
      P[i][j]=pdu*P[i][j]+puu*P[i][j+1]+pdd*P[i+1][j]+
  pud*P[i+1][j+1];
      if (am)
  {
    P[i][j]= MAX(iv[2*i+k][2*j+k],P[i][j]);
  }
    }
```

```
  }
    }
  /*Deltas*/
  MOD_OPT(Delta_Operator)(u1,d1,u2,d2,s1,s2,P[0][1],P[1][1]
    ,P[0][0],P[1][0],ptdelta1,ptdelta2);

  /*First Time Step*/
  P[0][0]=pdu*P[0][0]+puu*P[0][1]+pdd*P[1][0]+pud*P[1][1];;
  if (am)
    P[0][0]=MAX(iv[N][N],P[0][0]);
  /*Price*/
  *ptprice=P[0][0];

  /*Memory desallocation*/
  for (i=0;i<2*N+1;i++)
    free(iv[i]);
  free(iv);

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

  return OK;
}

int CALC(TR_BoyleEvnineGibbs)(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 BoyleEvnineGibbs_88(ptOpt->EuOrAm.Val.V_BOOL,pt
    Mod->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->Res[0].Val.V_
    DOUBLE),&(Met->Res[1].Val.V_DOUBLE),&(Met->Res[2].Val.V_DOUBLE)
    );
}

static int CHK_OPT(TR_BoyleEvnineGibbs)(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(TR_BoyleEvnineGibbs)=
{
  "TR_BoyleEvnineGibbs",
  {{"StepNumber",INT2,{100},ALLOW},{" ",PREMIA_NULLTYPE,{0}
    ,FORBID}},
  CALC(TR_BoyleEvnineGibbs),
  {{"Price",DOUBLE,{100},FORBID},{"Delta1",DOUBLE,{100},FO
    RBID} ,{"Delta2",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_BoyleEvnineGibbs),
  CHK_tree,
  MET(Init)
};
```

References