Help
```
#include <stdlib.h>
#include  "bs2d_std2d.h"
#include "error_msg.h"
#include "enums.h"

static double *Mesh=NULL, *Path=NULL, *Price=NULL, *VectIn
    vMeshDensity=NULL;
static double *Aux_BS_TD_1=NULL, *Aux_BS_TD_2=NULL, *InvSi
    gma=NULL;
static double Norm_BS_TD,DetInvSigma;
static double *AuxBS=NULL,*Sigma=NULL,*Aux_Stock=NULL;

static int BrGl_Allocation(long AL_Mesh_Size,int OP_Exercis
    e_Dates, int BS_Dimension)
{
  if (Mesh==NULL)
    Mesh= malloc(AL_Mesh_Size*OP_Exercise_Dates*BS_Dimensio
    n*sizeof(double));
  if (Mesh==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Price==NULL)
    Price= malloc(AL_Mesh_Size*OP_Exercise_Dates*sizeof(
    double));
  if (Price==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Path==NULL)
    Path= malloc(OP_Exercise_Dates*BS_Dimension*sizeof(
    double));

  if (Path==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (VectInvMeshDensity==NULL)
    VectInvMeshDensity= malloc(AL_Mesh_Size*sizeof(double))
    ;

  if (VectInvMeshDensity==NULL)
    return MEMORY_ALLOCATION_FAILURE;
```

```
  if (Aux_BS_TD_1==NULL)
    Aux_BS_TD_1= malloc(BS_Dimension*sizeof(double));
  if (Aux_BS_TD_1==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Aux_BS_TD_2==NULL)
    Aux_BS_TD_2= malloc(BS_Dimension*sizeof(double));
  if (Aux_BS_TD_2==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (InvSigma==NULL)
    InvSigma= malloc(BS_Dimension*BS_Dimension*sizeof(
    double));
  if (InvSigma==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Sigma==NULL)
    Sigma= malloc(BS_Dimension*BS_Dimension*sizeof(double))
    ;
  if (Sigma==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (AuxBS==NULL)
    AuxBS= malloc(BS_Dimension*sizeof(double));
  if (AuxBS==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Aux_Stock==NULL)
    Aux_Stock= malloc(BS_Dimension*sizeof(double));
  if (Aux_Stock==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  return OK;
}

static void Brod_Liberation()
{
  if (Mesh!=NULL) {
    free(Mesh);
    Mesh=NULL;
```

```
}

if (Price!=NULL) {
  free(Price);
  Price=NULL;
}

if (Path!=NULL) {
  free(Path);
  Path=NULL;
}

if (VectInvMeshDensity!=NULL) {
  free(VectInvMeshDensity);
  VectInvMeshDensity=NULL;
}

if (Aux_BS_TD_1!=NULL) {
  free(Aux_BS_TD_1);
  Aux_BS_TD_1=NULL;
}

if (Aux_BS_TD_2!=NULL) {
  free(Aux_BS_TD_2);
  Aux_BS_TD_2=NULL;
}

if (InvSigma!=NULL) {
  free(InvSigma);
  InvSigma=NULL;
}
if (Aux_Stock!=NULL) {
  free(Aux_Stock);
  Aux_Stock=NULL;
}

if (AuxBS!=NULL) {
  free(AuxBS);
  AuxBS=NULL;
}
```

```
  if (Sigma!=NULL) {
    free(Sigma);
    Sigma=NULL;
  }
}

static double Discount(double Time, double BS_Interest_Ra
    te)
{
  return exp(-BS_Interest_Rate*Time);
}

static void BS_Transition_Allocation(int BS_Dimension,
    double Step)
{
  int i;
  PnlMat InvSigma_wrap, Sigma_wrap;

  Sigma_wrap = pnl_mat_wrap_array (Sigma, BS_Dimension, BS_
    Dimension);
  InvSigma_wrap = pnl_mat_wrap_array (InvSigma, BS_Dimensio
    n, BS_Dimension);
  pnl_mat_inverse (&InvSigma_wrap, &Sigma_wrap);

  /* determinant of InvSigma */
  DetInvSigma=1;
  for (i=0;i<BS_Dimension;i++)
    DetInvSigma*=InvSigma[i*BS_Dimension+i];

  Norm_BS_TD=exp(-BS_Dimension*0.5*log(2.*M_PI*Step));
}

/*Black-Sholes Conditional Density function knowing Z*/
static double BS_TD(double *X, double *Z, int BS_Dimension,
     double Step)
{
  int i,j;
  double aux1,aux2;

  for (i=0;i<BS_Dimension;i++){
    Aux_BS_TD_1[i]=log(Z[i]/X[i])+Step*AuxBS[i];
```

```
  }
  aux1=Z[0];
  for (i=1;i<BS_Dimension;i++){
    aux1*=Z[i];
  }
  if (aux1==0){
    return -1;
  }
  else {
    for (i=0;i<BS_Dimension;i++){
      Aux_BS_TD_2[i]=0;
      for (j=0;j<=i;j++){
  Aux_BS_TD_2[i]+=InvSigma[i*BS_Dimension+j]*Aux_BS_TD_1[
    j];
      }
    }
    aux2=0;
    for (i=0;i<BS_Dimension;i++){
      aux2+=Aux_BS_TD_2[i]*Aux_BS_TD_2[i];
    }
    aux2=exp(-aux2/(2.*Step));
    return Norm_BS_TD*DetInvSigma*aux2/aux1;
  }
}

static double MeshDensity(int Time, double *Stock, int OP_
    Exercise_Dates, int AL_Mesh_Size,int BS_Dimension, double *
    BS_Spot, double Step)
{
  long k;
  double aux=0;

  if (Time>1){
    for (k=0;k<AL_Mesh_Size;k++)
      aux+=BS_TD(Mesh+k*OP_Exercise_Dates*BS_Dimension+(
    Time-1)*BS_Dimension,Stock,BS_Dimension,Step);
    return aux/(double)AL_Mesh_Size;
  } else {
    return BS_TD(BS_Spot,Stock,BS_Dimension,Step);
  }
}
```

```
static double Weight(int Time, double *iStock, double *jS
    tock, int j, int BS_Dimension,
         double Step)
{
  if (Time>0){
    return BS_TD(iStock,jStock,BS_Dimension,Step)*VectInvM
    eshDensity[j];
  } else {
    return 1.;
  }
}

static void BS_Forward_Step(int generator,double *Stock,
    double *Initial_Stock, int BS_Dimension,double Step,double Sqrt_
    Step)
{
  int j,k;
  double Aux;

  for (j=0;j<BS_Dimension;j++){
    Aux_Stock[j]=Sqrt_Step*pnl_rand_normal(generator);

  }
  for (j=0;j<BS_Dimension;j++){
    Aux=0.;
    for (k=0;k<=j;k++){
      Aux+=Sigma[j*BS_Dimension+k]*Aux_Stock[k];
    }
    Aux-=Step*AuxBS[j];
    Stock[j]=Initial_Stock[j]*exp(Aux);
  }
}

static void InitMesh(int generator,int AL_Mesh_Size, int
    BS_Dimension, double *BS_Spot,int OP_Exercise_Dates, double
    Step, double Sqrt_Step)
{
  int j,k, aux;
```

```
  for (k=0;k<AL_Mesh_Size;k++){
    BS_Forward_Step(generator,Mesh+k*OP_Exercise_Dates*BS_
    Dimension+BS_Dimension,BS_Spot,BS_Dimension,Step,Sqrt_Step);
  }
  for (j=2;j<OP_Exercise_Dates;j++){
    for (k=0;k<AL_Mesh_Size;k++){

      aux=(int) (AL_Mesh_Size*pnl_rand_uni(generator));

      BS_Forward_Step(generator,Mesh+k*OP_Exercise_Dates*
    BS_Dimension+j*BS_Dimension,Mesh+aux*OP_Exercise_Dates*BS_
    Dimension+(j-1)*BS_Dimension,BS_Dimension,Step,Sqrt_Step);
    }
  }
}

static void BrGl(double *AL_Price,
     long AL_MonteCarlo_Iterations,
     NumFunc_2 *p, int AL_Mesh_Size,
     int AL_ShuttingDown,
     int generator,
     int OP_Exercise_Dates,
     double *BS_Spot,
     double BS_Maturity,
     double BS_Interest_Rate,
     double *BS_Dividend_Rate,
     double *BS_Volatility,
     int gj_flag)
{
  double aux,Step,Sqrt_Step,DiscountStep;
  long i,j,k;
  int l;
  double AL_BPrice,AL_FPrice;
  int BS_Dimension=2;

  AL_BPrice=0.;
  AL_FPrice=0.;
  Step=BS_Maturity/(double)(OP_Exercise_Dates-1);
  Sqrt_Step=sqrt(Step);
  DiscountStep=exp(-BS_Interest_Rate*Step);
```

```
/*Memory Allocation*/
BrGl_Allocation(AL_Mesh_Size,OP_Exercise_Dates,BS_Dimens
  ion);

/*Black-Sholes initalization parameters*/
Sigma[0]=BS_Volatility[0];
Sigma[1]=BS_Volatility[1];
Sigma[2]=BS_Volatility[2];
Sigma[3]=BS_Volatility[3];
AuxBS[0]=0.5*(SQR(Sigma[0])+SQR(Sigma[1]))-BS_Interest_Ra
  te+BS_Dividend_Rate[0];
AuxBS[1]=0.5*(SQR(Sigma[2])+SQR(Sigma[3]))-BS_Interest_Ra
  te+BS_Dividend_Rate[1];
BS_Transition_Allocation(BS_Dimension,Step);

/*Initialization of the mesh*/
InitMesh(generator,AL_Mesh_Size,BS_Dimension,BS_Spot,OP_
  Exercise_Dates,Step,Sqrt_Step);


for (i=0;i<AL_Mesh_Size;i++)
  Price[i*OP_Exercise_Dates+OP_Exercise_Dates-1]=0.;

/*Dynamical programing: Backward Price */
for (j=OP_Exercise_Dates-2;j>=1;j--){
  for (i=0;i<AL_Mesh_Size;i++){
    VectInvMeshDensity[i]=1./MeshDensity(j+1,Mesh+i*OP_
  Exercise_Dates*BS_Dimension+(j+1)*BS_Dimension,OP_Exercise_
  Dates,AL_Mesh_Size,BS_Dimension,BS_Spot,Step);

  }
  for (i=0;i<AL_Mesh_Size;i++){
    aux=0;
    for (k=0;k<AL_Mesh_Size;k++){

/*Payoff control variate*/
aux+=(Price[k*OP_Exercise_Dates+j+1]+
      (p->Compute) (p->Par,*(Mesh+k*OP_Exercise_Dates*
  BS_Dimension+(j+1)*BS_Dimension),*(Mesh+k*OP_Exercise_Dates*
  BS_Dimension+(j+1)*BS_Dimension+1)))*Weight(j,Mesh+i*OP_Exe
  rcise_Dates*BS_Dimension+j*BS_Dimension,Mesh+k*OP_Exercise_
```

```
  Dates*BS_Dimension+(j+1)*BS_Dimension,k,BS_Dimension,Step);
    }
    aux*=DiscountStep/(double)AL_Mesh_Size;
    aux-=(p->Compute)(p->Par,*(Mesh+i*OP_Exercise_Dates*
  BS_Dimension+j*BS_Dimension),
    *(Mesh+i*OP_Exercise_Dates*BS_Dimension+j*BS_Dim
  ension+1));
    Price[i*OP_Exercise_Dates+j]=MAX(0,aux);
  }
}

aux=0;
for (i=0;i<AL_Mesh_Size;i++){
  aux+=Price[i*OP_Exercise_Dates+1]+(p->Compute) (p->Par,
  *(Mesh+i*OP_Exercise_Dates*BS_Dimension+BS_Dimension),*(
  Mesh+i*OP_Exercise_Dates*BS_Dimension+BS_Dimension+1));
}

/*Backward Price*/
if(!gj_flag)
  AL_BPrice=MAX((p->Compute) (p->Par,*(BS_Spot),*(BS_Spo
  t+1)),DiscountStep*aux/(double)AL_Mesh_Size);
else
  AL_BPrice=DiscountStep*aux/(double)AL_Mesh_Size;

/* Forward Price */
AL_FPrice=0;
for(i=0;i<AL_MonteCarlo_Iterations;i++){

  for (l=0;l<BS_Dimension;l++){
    Path[l]=BS_Spot[l];
  }

  j=0;
  do {
    aux=0;
    for (k=0;k<AL_Mesh_Size;k++){
aux+=(Price[k*OP_Exercise_Dates+j+1]+(p->Compute) (p->
  Par,*(Mesh+k*OP_Exercise_Dates*BS_Dimension+(j+1)*BS_Dimensio
  n),*(Mesh+k*OP_Exercise_Dates*BS_Dimension+(j+1)*BS_Dimens
  ion+1)))*Weight(j,Path+j*BS_Dimension,Mesh+k*OP_Exercise_Da
```

```
    tes*BS_Dimension+(j+1)*BS_Dimension,k,BS_Dimension,Step);
      }
      aux*=DiscountStep/(double)AL_Mesh_Size;
      aux-=(p->Compute)(p->Par,*(Path+j*BS_Dimension),*(
    Path+j*BS_Dimension+1));

      j++;
      BS_Forward_Step(generator,Path+j*BS_Dimension,Path+(
    j-1)*BS_Dimension,BS_Dimension,Step,Sqrt_Step);
    }
    while ((0<aux)&&(j<OP_Exercise_Dates-1));
    AL_FPrice+=Discount((double)(j)*Step,BS_Interest_Rate)*
    (p->Compute) (p->Par,*(Path+(j)*BS_Dimension),*(Path+(j)*
    BS_Dimension+1));
  }
  AL_FPrice/=(double)AL_MonteCarlo_Iterations;

  /*Price = Mean of Forward and Backward Price*/
  *AL_Price=0.5*(AL_FPrice+AL_BPrice);

  /*Memory Disallocation*/
  if (AL_ShuttingDown){
    Brod_Liberation();
  }
}


static int MCBroadieGlassermann2D(double s1, double s2,
    NumFunc_2  *p, double t, double r, double divid1, double divid2,
    double sigma1, double sigma2, double rho, long N, int    generator,  double
    double *ptprice, double *ptdelta1, double *ptdelta2)
{

  double p1,p2,p3;
  int simulation_dim= 1,fermeture=1,init_mc;
  double s_vector[2];
  double s_vector_plus1[2],s_vector_plus2[2];
  double sigma[4];
  double divid[2];
```

```
/* Covariance Matrix */
/* Coefficients of the matrix A such that A(tA)=Gamma */
sigma[0]= sigma1;
sigma[1]= 0.0;
sigma[2]= rho*sigma2;
sigma[3]= sigma2*sqrt(1.0-SQR(rho));

/*Initialisation*/
s_vector[0]=s1;
s_vector[1]=s2;
s_vector_plus1[0]=s1*(1.+inc);
s_vector_plus1[1]=s2;
s_vector_plus2[0]=s1;
s_vector_plus2[1]=s2*(1.+inc);
divid[0]=divid1;
divid[1]=divid2;

/*MC sampling*/
init_mc= pnl_rand_init(generator,simulation_dim,N);

/* Test after initialization for the generator */
if(init_mc == OK)
  {

    /*Geske-Johnson Formulae*/
    if (exercise_date_number==0) {
BrGl(&p3,N,p,mesh_size,fermeture,generator,4,s_vector,t,
  r,divid,sigma,1);
BrGl(&p2,N,p,mesh_size,fermeture,generator,3,s_vector,t,
  r,divid,sigma,1);
BrGl(&p1,N,p,mesh_size,fermeture,generator,2,s_vector,t,
  r,divid,sigma,1);
*ptprice=p3+7./2.*(p3-p2)-(p2-p1)/2.;
    } else {
BrGl(ptprice,N,p,mesh_size,fermeture,generator,exercise_
  date_number,s_vector,t,r,divid,sigma,0);
    }

    /*Delta*/
    if (exercise_date_number==0) {
BrGl(&p1,N,p,mesh_size,fermeture,generator,2,s_vector_pl
```

```
    us1,t,r,divid,sigma,1);
  BrGl(&p2,N,p,mesh_size,fermeture,generator,3,s_vector_pl
    us1,t,r,divid,sigma,1);
  BrGl(&p3,N,p,mesh_size,fermeture,generator,4,s_vector_pl
    us1,t,r,divid,sigma,1);
  *ptdelta1=((p3+7./2.*(p3-p2)-(p2-p1)/2.)-*ptprice)/(s1*
    inc);
  BrGl(&p1,N,p,mesh_size,fermeture,generator,2,s_vector_pl
    us2,t,r,divid,sigma,1);
  BrGl(&p2,N,p,mesh_size,fermeture,generator,3,s_vector_pl
    us2,t,r,divid,sigma,1);
  BrGl(&p3,N,p,mesh_size,fermeture,generator,4,s_vector_pl
    us2,t,r,divid,sigma,1);
  *ptdelta2=((p3+7./2.*(p3-p2)-(p2-p1)/2.)-*ptprice)/(s2*
    inc);

      } else {
  BrGl(&p1,N,p,mesh_size,fermeture,generator,exercise_da
    te_number,s_vector_plus1,t,r,divid,sigma,0);
  *ptdelta1=(p1-*ptprice)/(s1*inc);
  BrGl(&p2,N,p,mesh_size,fermeture,generator,exercise_da
    te_number,s_vector_plus2,t,r,divid,sigma,0);
  *ptdelta2=(p2-*ptprice)/(s2*inc);
      }
    }
  return init_mc;
}

int CALC(MC_BroadieGlassermann2D)(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->Divid1.Val.V_DOUBLE/100.);

  return MCBroadieGlassermann2D(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_LONG,
        Met->Par[1].Val.V_ENUM.value,
        Met->Par[2].Val.V_PDOUBLE,
        Met->Par[3].Val.V_INT,
        Met->Par[4].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(MC_BroadieGlassermann2D)(void *Opt, voi
    d *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

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

}



static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;
      Met->Par[0].Val.V_LONG=10000;
      Met->Par[1].Val.V_ENUM.value=0;
      Met->Par[1].Val.V_ENUM.members=&PremiaEnumMCRNGs;
```

```
      Met->Par[2].Val.V_PDOUBLE=0.1;
      Met->Par[3].Val.V_INT=200;
      Met->Par[4].Val.V_INT=20;

    }

  return OK;
}

PricingMethod MET(MC_BroadieGlassermann2D)=
{
  "MC_BroadieGlassermann2d",
  {{"N iterations",LONG,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"Delta Increment Rel",PDOUBLE,{100},ALLOW},
   {"Mesh Size",INT,{100},ALLOW},
   {"Number of Exercise Dates (0->Geske Johnson Formulae)",
    INT,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_BroadieGlassermann2D),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta1",DOUBLE,{100},FORBID} ,{"Delta2",DOUBLE,{100},
    FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_BroadieGlassermann2D),
  CHK_mc,
  MET(Init)
};
```

References