Help

```
/* Broadie & Glasserman algorithm (stochastic mesh)*/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "bsnd_stdnd.h"
#include "math/linsys.h"
#include "pnl/pnl_basis.h"
#include "black.h"
#include "optype.h"
#include "enums.h"
#include "var.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_matrix.h"

static double *Mesh=NULL, *Path=NULL, *Price=NULL, *VectIn
    vMeshDensity=NULL;

static int BrGl_Allocation(long AL_Mesh_Size,
                           int OP_Exercise_Dates, int BS_
    Dimension)
{
  if (Mesh==NULL) Mesh=(double*)malloc(AL_Mesh_Size*OP_Exe
    rcise_Dates*BS_Dimension*sizeof(double));
  if (Mesh==NULL) return MEMORY_ALLOCATION_FAILURE;
  if (Price==NULL)  Price=(double*)malloc(AL_Mesh_Size*OP_
    Exercise_Dates*sizeof(double));
  if (Price==NULL) return MEMORY_ALLOCATION_FAILURE;
  if (Path==NULL)   Path=(double*)malloc(OP_Exercise_Dates*
    BS_Dimension*sizeof(double));
  if (Path==NULL) return MEMORY_ALLOCATION_FAILURE;
  if (VectInvMeshDensity==NULL) VectInvMeshDensity=(double*
    )malloc(OP_Exercise_Dates*AL_Mesh_Size*sizeof(double));
  if (VectInvMeshDensity==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; }
}

static double MeshDensity(int Time, double *Stock, int OP_
    Exercise_Dates, int AL_Mesh_Size,
                          int BS_Dimension, double *BS_Spo
    t, double Step)
{
  long k;
  double aux=0;
  /*density function of the mesh law generator */
  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, int AL_Mesh_Size)
{
  /*computation of the weight between the vectors iStock an
    d jStock*/
  if (Time>0)
    return BS_TD(iStock,jStock,BS_Dimension,Step)*VectInvM
    eshDensity[Time*AL_Mesh_Size+j];
  else
    return 1.;

}

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

  /*mesh initialization; see the documentation*/
  for (k=0;k<AL_Mesh_Size;k++)
    BS_Forward_Step(Mesh+k*OP_Exercise_Dates*BS_Dimension+
    BS_Dimension,BS_Spot,BS_Dimension,Step,Sqrt_Step,     generator);

  for (j=2;j<OP_Exercise_Dates;j++)
    {
      for (k=0;k<AL_Mesh_Size;k++)
        {
          aux=(int)(pnl_rand_uni(generator)*AL_Mesh_Size);
          BS_Forward_Step(Mesh+k*OP_Exercise_Dates*BS_Dim
    ension+j*BS_Dimension,Mesh+aux*OP_Exercise_Dates*BS_Dimensio
    n+(j-1)*BS_Dimension,BS_Dimension,Step,Sqrt_Step,     generator);
        }
    }
}


static void Close()
{
  /*memory liberation*/
  Brod_Liberation();
  BS_Transition_Liberation();
  End_BS();
}

/*see the documentation for the parameters meaning*/
static int BrGl(PnlVect *BS_Spot,
                NumFunc_nd *p,
                double OP_Maturity,
                double BS_Interest_Rate,
                PnlVect *BS_Dividend_Rate,
                PnlVect *BS_Volatility,
                double *BS_Correlation,
                long AL_MonteCarlo_Iterations,
                int generator,
                int AL_Mesh_Size,
                int OP_Exercise_Dates,
                double *AL_FPrice,
                double *AL_BPrice)
{
  double aux,Step,Sqrt_Step,DiscountStep;
  long i,j,k, init_mc;
  int l;
  /* double AL_FPrice, AL_BPrice; */
  int BS_Dimension =  BS_Spot->size;
  PnlVect VMesh;
  VMesh.size =  BS_Dimension;

  /* MC sampling */
  init_mc= pnl_rand_init(generator, BS_Dimension, AL_
    MonteCarlo_Iterations);
    /* Test after initialization for the generator */
  if(init_mc != OK)  return init_mc;

  /*time step*/
  Step=OP_Maturity/(double)(OP_Exercise_Dates-1);
  Sqrt_Step=sqrt(Step);
  /*discounting factor for a time step*/
  DiscountStep=exp(-BS_Interest_Rate*Step);

  Init_BS(BS_Dimension,BS_Volatility->array,
          BS_Correlation,BS_Interest_Rate,BS_Dividend_Rate-
    >array);
  /*memory allocation of the BlackScholes variables*/
  BS_Transition_Allocation(BS_Dimension,Step);
  /*memory allocation of the algorithm's variables*/
  BrGl_Allocation(AL_Mesh_Size,OP_Exercise_Dates,BS_Dimens
    ion);

  /*initialization of the mesh*/
  InitMesh(AL_Mesh_Size,BS_Dimension,BS_Spot->array,OP_Exe
    rcise_Dates,Step,Sqrt_Step, generator);

  /* Backward Price */
  /*partial computation of the weights*/
```

```
for (j=OP_Exercise_Dates-2;j>=1;j--){
  for (i=0;i<AL_Mesh_Size;i++){
    VectInvMeshDensity[j*AL_Mesh_Size+i]=1./MeshDensity(
  j+1,Mesh+i*OP_Exercise_Dates*BS_Dimension+(j+1)*BS_Dimensio
  n,OP_Exercise_Dates,AL_Mesh_Size,BS_Dimension,BS_Spot->arra
  y,Step);
  }
}
/*initialization of the mesh prices at the maturity*/
for (i=0;i<AL_Mesh_Size;i++)
  {
    VMesh.array=Mesh+i*OP_Exercise_Dates*BS_Dimension+(
  OP_Exercise_Dates-1)*BS_Dimension;
    Price[i*OP_Exercise_Dates+OP_Exercise_Dates-1]=p->
  Compute(p->Par, &VMesh);
  }

/*dynamical programming algorithm*/
for (j=OP_Exercise_Dates-2;j>=1;j--){
  for (i=0;i<AL_Mesh_Size;i++){
    aux=0;
    /*approximation of the conditionnal expectation*/
    for (k=0;k<AL_Mesh_Size;k++){
      aux+=Price[k*OP_Exercise_Dates+j+1]*Weight(j,Mesh+
  i*OP_Exercise_Dates*BS_Dimension+j*BS_Dimension,Mesh+k*OP_
  Exercise_Dates*BS_Dimension+(j+1)*BS_Dimension,k,BS_Dimensio
  n,Step,AL_Mesh_Size);
    }
    aux*=DiscountStep/(double)AL_Mesh_Size;
    /*exercise decision*/
    VMesh.array = Mesh+i*OP_Exercise_Dates*BS_Dimension+
  j*BS_Dimension;
    Price[i*OP_Exercise_Dates+j]=MAX(p->Compute(p->Par, &
  VMesh),aux);
  }
}

aux=0;
for (i=0;i<AL_Mesh_Size;i++){
  aux+=Price[i*OP_Exercise_Dates+1];
}
```

```
/*output backward price*/
*AL_BPrice=MAX(p->Compute(p->Par,BS_Spot),DiscountStep*au
  x/(double)AL_Mesh_Size);

/* Forward Price */
if (*AL_BPrice==p->Compute(p->Par,BS_Spot))
  *AL_FPrice=*AL_BPrice;
else
  {
    *AL_FPrice=0.;
    for(i=0;i<AL_MonteCarlo_Iterations;i++){
      /*BlackScholes spot*/
      for (l=0;l<BS_Dimension;l++)
        Path[l]=BS_Spot->array[l];

      j=0;
      /*optimal stopping of a BlackScholes path*/
      do {
        j++;
        aux=0;
        /*approxiamtion of the continuation value*/
        for (k=0;k<AL_Mesh_Size;k++){
          aux+=(Price[k*OP_Exercise_Dates+j+1])*Weight(j,
  Path+j*BS_Dimension,Mesh+k*OP_Exercise_Dates*BS_Dimension+(j+1
  )*BS_Dimension,k,BS_Dimension,Step,AL_Mesh_Size);
        }
        aux*=DiscountStep/(double)AL_Mesh_Size;
        VMesh.array=Path+j*BS_Dimension;
        aux-=p->Compute(p->Par, &VMesh);
        /*BlackScholes stock increment*/
        BS_Forward_Step(Path+j*BS_Dimension,Path+(j-1)*
  BS_Dimension,BS_Dimension,Step,Sqrt_Step, generator);

      }
      while ((0<aux)&&(j<OP_Exercise_Dates-1));
      /*MonteCarlo formulae for the forward price*/
       VMesh.array=Path+j*BS_Dimension;
      *AL_FPrice+=Discount((double)(j)*Step,BS_Interest_
  Rate)*p->Compute(p->Par, &VMesh);
    }
    /*output forward price*/
```

```
      *AL_FPrice/=(double)AL_MonteCarlo_Iterations;
    }
  Close();
  return OK;
}

int CALC(MC_BroadieGlassermannND)(void *Opt, void *Mod,
    PricingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r;
  double *BS_cor;
  int i, res;
  PnlVect *divid = pnl_vect_create(ptMod->Size.Val.V_PINT);
  PnlVect *spot, *sig;

  spot = pnl_vect_compact_to_pnl_vect (ptMod->S0.Val.V_PNLV
    ECTCOMPACT);
  sig = pnl_vect_compact_to_pnl_vect (ptMod->Sigma.Val.V_PN
    LVECTCOMPACT);

  for(i=0; i<ptMod->Size.Val.V_PINT; i++)
    pnl_vect_set (divid, i,
           log(1.+ pnl_vect_compact_get (ptMod->Divid.Val.
    V_PNLVECTCOMPACT, i)/100.));

  r= log(1.+ptMod->R.Val.V_DOUBLE/100.);

  if ((BS_cor = malloc(ptMod->Size.Val.V_PINT*ptMod->Size.
    Val.V_PINT*sizeof(double)))==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  for(i=0; i<ptMod->Size.Val.V_PINT*ptMod->Size.Val.V_PINT;
     i++)
    BS_cor[i]= ptMod->Rho.Val.V_DOUBLE;
  for(i=0; i<ptMod->Size.Val.V_PINT; i++)
    BS_cor[i*ptMod->Size.Val.V_PINT+i]= 1.0;

  res=BrGl(spot,
           ptOpt->PayOff.Val.V_NUMFUNC_ND,
           ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DATE,
           r, divid, sig,
           BS_cor,
           Met->Par[0].Val.V_LONG,
           Met->Par[1].Val.V_ENUM.value,
           Met->Par[2].Val.V_INT,
           Met->Par[3].Val.V_INT,
           &(Met->Res[0].Val.V_DOUBLE),
           &(Met->Res[1].Val.V_DOUBLE));
  pnl_vect_free(&divid);
  free(BS_cor);
  pnl_vect_free (&spot);
  pnl_vect_free (&sig);

  return res;
}

static int CHK_OPT(MC_BroadieGlassermannND)(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_INT=200;
      Met->Par[3].Val.V_INT=20;
    }

  return OK;
}

PricingMethod MET(MC_BroadieGlassermannND)=
{
  "MC_BroadieGlassermann_ND",
  {{"N iterations",LONG,{100},ALLOW},
   {"RandomGenerator",ENUM,{0},ALLOW},
   {"Mesh Size",INT,{100},ALLOW},
   {"Number of Exercise Dates",INT,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_BroadieGlassermannND),
  {{"Price Forward",DOUBLE,{100},FORBID},{"Price Backward",
    DOUBLE,{100},FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_BroadieGlassermannND),
  CHK_mc,
  MET(Init)
};
```

References