Help

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

static double *Q=NULL, *Weights=NULL, *Trans=NULL, *Price=
    NULL;
static double *Aux_Path=NULL,*Aux_Stock=NULL,*Aux_BS=NULL;
static double *Sigma=NULL;
static int *Path_Int=NULL;

static int RaQ_Allocation(int AL_T_Size, int BS_Dimension,
        int OP_Exercice_Dates)
{
  if (Q==NULL)
    Q= malloc(AL_T_Size*OP_Exercice_Dates*BS_Dimension*size
    of(double));
  if (Q==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Trans==NULL)
    Trans= malloc(OP_Exercice_Dates*AL_T_Size*AL_T_Size*si
    zeof(double));
  if (Trans==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Weights==NULL)
    Weights= malloc(OP_Exercice_Dates*AL_T_Size*sizeof(
    double));
  if (Weights==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Price==NULL)
    Price= malloc(OP_Exercice_Dates*AL_T_Size*sizeof(
    double));
  if (Price==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  if (Aux_Path==NULL)
```

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

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

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

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

  if (Path_Int==NULL)
    Path_Int= malloc(OP_Exercice_Dates*sizeof(int));
  if (Path_Int==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  return OK;
}

static void RaQ_Liberation()
{
  if (Q!=NULL) {
    free(Q);
    Q=NULL;
  }
  if (Trans!=NULL) {
    free(Trans);
    Trans=NULL;
  }
  if (Weights!=NULL) {
    free(Weights);
```

```
    Weights=NULL;
  }
  if (Price!=NULL) {
    free(Price);
    Price=NULL;
  }
  if (Aux_Path!=NULL) {
    free(Aux_Path);
    Aux_Path=NULL;
  }
  if (Aux_Stock!=NULL) {
    free(Aux_Stock);
    Aux_Stock=NULL;
  }

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

  if (Sigma!=NULL) {
    free(Sigma);
    Sigma=NULL;
  }
  if (Path_Int!=NULL) {
    free(Path_Int);
    Path_Int=NULL;
  }
  return;
}

static int NearestCell(int Time, int AL_T_Size, long OP_Em
    BS_Di, int BS_Dimension)
{
  int j,k,l=0;
  double  min=DBL_MAX,aux,auxnorm;
  for (j=0;j<AL_T_Size;j++){
    aux=0;
    for (k=0;k<BS_Dimension;k++){
      auxnorm=Aux_Path[Time*BS_Dimension+k]-
  Q[(long)j*OP_EmBS_Di+Time*BS_Dimension+k];
```

```
      aux+=auxnorm*auxnorm;
    }
    if (min>aux){
      min=aux;
      l=j;
    }
  }
  return l;
}
static void ForwardPath(double *Path, double *Initial_Stock
    , int Initial_Time,int Number_Dates,int generator,int BS_
    Dimension, double Step, double Sqrt_Step)
{
  int i,j,k;
  double aux;
  double *SigmapjmBS_Dimensionpk;

  for (j=0;j<BS_Dimension;j++) Path[Initial_Time*BS_Dimens
    ion+j]=Initial_Stock[j];

  for (i=Initial_Time+1;i<Initial_Time+Number_Dates;i++){
    for (j=0;j<BS_Dimension;j++){
      Aux_Stock[j]=Sqrt_Step*pnl_rand_normal(generator);
    }
    SigmapjmBS_Dimensionpk=Sigma;


    for (j=0;j<BS_Dimension;j++){
      aux=0.;
      for (k=0;k<=j;k++){
        aux+=(*SigmapjmBS_Dimensionpk)*Aux_Stock[k];
  SigmapjmBS_Dimensionpk++;
      }
      SigmapjmBS_Dimensionpk+=BS_Dimension-j-1;
      aux-=Step*Aux_BS[j];
      Path[i*BS_Dimension+j]=Path[(i-1)*BS_Dimension+j]*exp
    (aux);
    }
  }
}
```

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

static void Init_Tesselations(long AL_MonteCarlo_Iterations
    , int AL_T_Size,int OP_Exercice_Dates,int generator,int
    BS_Dimension, double *BS_Spot,double Step, double Sqrt_Step)
{
  int i,j,k,Vimoins,Vi;
  long l;
  long OP_ExmBS_Di=(long)OP_Exercice_Dates*BS_Dimension;

  /* Random Quantizers */
  for (i=0;i<AL_T_Size;i++)
    ForwardPath(Q+i*OP_Exercice_Dates*BS_Dimension,BS_Spot,
    0,OP_Exercice_Dates,
    generator,BS_Dimension,Step,Sqrt_Step);

  /* Weights and Transitions */
  for (i=0;i<OP_Exercice_Dates;i++)
    for (j=0;j<AL_T_Size;j++)
      Weights[i*AL_T_Size+j]=0;

  for (i=0;i<OP_Exercice_Dates;i++)
    for (j=0;j<AL_T_Size;j++)
      for (k=0;k<AL_T_Size;k++)
  Trans[i*AL_T_Size*AL_T_Size+j*AL_T_Size+k]=0;

  for (l=0;l<AL_MonteCarlo_Iterations-AL_T_Size;l++){

    /*Black-Sholes Paths from time 0 to maturity*/
    ForwardPath(Aux_Path,BS_Spot,0,OP_Exercice_Dates,    generator,BS_Dimension,

    Vimoins=0;
    for (i=1;i<OP_Exercice_Dates;i++){
      Vi=NearestCell(i,AL_T_Size,OP_ExmBS_Di,BS_Dimension);
      Weights[i*AL_T_Size+Vi]+=1;
      Trans[i*AL_T_Size*AL_T_Size+Vimoins*AL_T_Size+Vi]+=1;
      Vimoins=Vi;
```

```
    }
  }
  Weights[0]=AL_MonteCarlo_Iterations-AL_T_Size;
  for (i=1;i<OP_Exercice_Dates;i++)
    for (j=0;j<AL_T_Size;j++)
      if (Weights[(i-1)*AL_T_Size+j]>0)
  for (k=0;k<AL_T_Size;k++)
    Trans[i*AL_T_Size*AL_T_Size+j*AL_T_Size+k]/=Weights[(
    i-1)*AL_T_Size+j];
}


static void RaQ(double *PrixDir,long MC_Iterations,NumFunc_
    2 *p,int size,int Fermeture,int generator,int exercise_da
    te_number,double *s_vector, double t, double r, double *div
    id, double *sigma,int gj_flag)
{
  int i,j,k,BS_Dimension=2;
  long l;
  double step,Sqrt_Step,DiscountStep,aux,AL_BPrice,AL_FPric
    e;


  *PrixDir=0.;
  step=t/(exercise_date_number-1.);
  Sqrt_Step=sqrt(step);
  DiscountStep=exp(-r*step);

  /*Memory Allocation*/
  RaQ_Allocation(size,BS_Dimension,exercise_date_number);

  /*Black-Sholes initalization parameters*/
  Sigma[0]=sigma[0];
  Sigma[1]=sigma[1];
  Sigma[2]=sigma[2];
  Sigma[3]=sigma[3];

  Aux_BS[0]=0.5*(SQR(sigma[0])+SQR(sigma[1]))-r+divid[0];
  Aux_BS[1]=0.5*(SQR(sigma[2])+SQR(sigma[3]))-r+divid[1];

  /* Cells Weights and Transitions probabilities   */
```

```
Init_Tesselations(MC_Iterations,size,exercise_date_numb
  er,generator,BS_Dimension,s_vector,step,Sqrt_Step);

for (i=0;i<size;i++)
  Price[(exercise_date_number-1)*size+i]=0;

/*  Dynamical programing  (backward price)*/
for (i=exercise_date_number-2;i>=1;i--) {
  for (j=0;j<size;j++){
    aux=0;

    /*Payoff control variate*/
    for (k=0;k<size;k++) {
aux+=(Price[(i+1)*size+k]+(p->Compute) (p->Par,*(Q+k*exe
  rcise_date_number*BS_Dimension+(i+1)*BS_Dimension),*(Q+k*
  exercise_date_number*BS_Dimension+(i+1)*BS_Dimension+1)))*
  Trans[(i+1)*size*size+j*size+k];
    }
    aux*=DiscountStep;
    aux-=(p->Compute) (p->Par,*(Q+j*exercise_date_number*
  BS_Dimension+i*BS_Dimension),*(Q+j*exercise_date_number*BS_
  Dimension+i*BS_Dimension+1));
    Price[i*size+j]=MAX(0.,aux);
  }
}

aux=0;
for (k=0;k<size;k++)
  aux+=(Price[size+k]+(p->Compute) (p->Par,*(Q+k*exercis
  e_date_number*BS_Dimension+BS_Dimension),*(Q+k*exercise_da
  te_number*BS_Dimension+BS_Dimension+1)))*Trans[size*size+k];

/*Backward Price*/
aux*=DiscountStep;
if(!gj_flag)
  AL_BPrice=MAX((p->Compute) (p->Par,s_vector[0],s_vector
  [1]),aux);
else AL_BPrice=aux;


/*  Forward price */
```

```
for (k=0;k<size;k++){
  Price[k]=AL_BPrice-(p->Compute)(p->Par,s_vector[0],s_
  vector[1]);
}
AL_FPrice=0.0;
for (j=0;j<size;j++){
  i=-1;
  do {
    i++;
  }
  while (0<Price[i*size+j]);

  AL_FPrice+=Discount((double)i*step,r)*(Price[i*size+j]+
           (p->Compute) (p->Par,*(Q+j*exercise_date_
  number*BS_Dimension+i*BS_Dimension),*(Q+j*exercise_date_numb
  er*BS_Dimension+i*BS_Dimension+1)));
}

for (l=0;l<MC_Iterations-size;l++){

  ForwardPath(Aux_Path,s_vector,0,exercise_date_number,    generator,BS_Dimens
  Path_Int[0]=0;
  for (i=1;i<exercise_date_number;i++){
    Path_Int[i]=NearestCell(i,size,exercise_date_number*
          BS_Dimension,BS_Dimension);
  }

  i=-1;
  do {
    i++;
  }
  while (0<Price[i*size+Path_Int[i]]);
  AL_FPrice+=Discount((double)i*step,r)*(Price[i*size+
  Path_Int[i]]+
           (p->Compute) (p->Par,*(Q+Path_Int[i]*exe
  rcise_date_number*BS_Dimension+i*BS_Dimension),*(Q+Path_Int
  [i]*exercise_date_number*BS_Dimension+i*BS_Dimension+1)));
}

AL_FPrice/=(double)MC_Iterations;
```

```
  /*Memory Disallocation*/
  if (Fermeture)
    RaQ_Liberation();

  /*Price=Forward Price*/
  *PrixDir=AL_FPrice;

  return;

}

static int MCRandomQuantization2D(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 i
    tion,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.+increment);
  s_vector_plus1[1]=s2;
  s_vector_plus2[0]=s1;
  s_vector_plus2[1]=s2*(1.+increment);
  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) {
RaQ(&p1,N,p,size_tesselation,fermeture,generator,2,s_vec
  tor,t,r,divid,sigma,1);
RaQ(&p2,N,p,size_tesselation,fermeture,generator,3,s_vec
  tor,t,r,divid,sigma,1);
RaQ(&p3,N,p,size_tesselation,fermeture,generator,4,s_vec
  tor,t,r,divid,sigma,1);
*ptprice=p3+7./2.*(p3-p2)-(p2-p1)/2.;
    } else {
RaQ(ptprice,N,p,size_tesselation,fermeture,generator,ex
  ercise_date_number,s_vector,t,r,divid,sigma,0);
    }

    /*Delta*/
    if (exercise_date_number==0){
RaQ(&p1,N,p,size_tesselation,fermeture,generator,2,s_vec
  tor_plus1,t,r,divid,sigma,1);
RaQ(&p2,N,p,size_tesselation,fermeture,generator,3,s_vec
  tor_plus1,t,r,divid,sigma,1);
RaQ(&p3,N,p,size_tesselation,fermeture,generator,4,s_vec
  tor_plus1,t,r,divid,sigma,1);

*ptdelta1=((p3+7./2.*(p3-p2)-(p2-p1)/2)-*ptprice)/(s1*
  increment);

RaQ(&p1,N,p,size_tesselation,fermeture,generator,2,s_vec
  tor_plus2,t,r,divid,sigma,1);
RaQ(&p2,N,p,size_tesselation,fermeture,generator,3,s_vec
  tor_plus2,t,r,divid,sigma,1);
RaQ(&p3,N,p,size_tesselation,fermeture,generator,4,s_vec
  tor_plus2,t,r,divid,sigma,1);
*ptdelta2=((p3+7./2.*(p3-p2)-(p2-p1)/2)-*ptprice)/(s2*
  increment);
```

```
      } else {
  RaQ(&p1,N,p,size_tesselation,fermeture,generator,exercis
    e_date_number,s_vector_plus1,t,r,divid,sigma,0);
  RaQ(&p2,N,p,size_tesselation,fermeture,generator,exercis
    e_date_number,s_vector_plus2,t,r,divid,sigma,0);
  *ptdelta1=(p1-*ptprice)/(s1*increment);
  *ptdelta2=(p2-*ptprice)/(s2*increment);
      }
    }
  return init_mc;
}

int CALC(MC_RandomQuantization2D)(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 MCRandomQuantization2D(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_RandomQuantization2D)(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=20;
      Met->Par[4].Val.V_INT=250;

    }
  return OK;
}

PricingMethod MET(MC_RandomQuantization2D)=
{
  "MC_RandomQuantization2d",
  {{"N iterations",LONG,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"Delta Increment Rel",PDOUBLE,{100},ALLOW},
   {"Number of Exercise Dates (0->Geske Johnson Formulae)",
    INT,{100},ALLOW},
   {"Tesselation Size",INT,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
```

```
  CALC(MC_RandomQuantization2D),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta1",DOUBLE,{100},FORBID} ,
   {"Delta2",DOUBLE,{100},FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_RandomQuantization2D),
   CHK_mc,
  MET(Init)
};
```

References