Help

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <float.h>

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

/* epsilon to detect if continuation value is reached */
#define EPS_CONT 0.0000001
#define PRECISION 1.0e-7 /*Precision for the localization
    of FD methods*/

static double *Grid=NULL,*Trans=NULL,*Delta=NULL;
static int *Succ=NULL,*TSize=NULL;

static double *Price=NULL,*BSQ=NULL;
static double machep = 0.000001;
static double beta_BS_Correlation, sigma_BS_Volatility, de
    lta_BS_Dividend_Rate;
static double beta_Basket_BS_Correlation, sigma_Basket_BS_
    Volatility, delta_Basket_BS_Dividend_Rate;

static int QOptStored_Allocation(int Dimension, int Nbpt)
{
  if (BSQ==NULL)BSQ=(double*)malloc(Nbpt*Dimension*sizeof(
    double));
  if (BSQ==NULL) return MEMORY_ALLOCATION_FAILURE;
  if (Price==NULL)  Price=(double*)malloc(Nbpt*sizeof(
```

```
    double));
  if (Price==NULL) return MEMORY_ALLOCATION_FAILURE;
  if (Delta==NULL) Delta=(double*)malloc(Nbpt*Dimension*si
    zeof(double));
  if (Delta==NULL) return MEMORY_ALLOCATION_FAILURE;
  return OK;
}

static void QOptStored_Liberation()
{
  if (BSQ!=NULL) { free(BSQ);  BSQ=NULL;}
  if (Grid!=NULL) { free(Grid); Grid=NULL; }
  if (TSize!=NULL) { free(TSize);  TSize=NULL; }
  if (Succ!=NULL) { free(Succ); Succ=NULL; }
  if (Trans!=NULL) { free(Trans); Trans=NULL; }
  if (Price!=NULL) { free(Price); Price=NULL;  }
  if (Delta!=NULL) { free(Delta); Delta=NULL; }
}

static int read_geom(double *MaxMaturity, int *MaxExerciseD
    ates, int *Dimension, int *Nbpt, char *name)
{
  int i,id,k,nvl;
  FILE *geomfp;
  double t;

  geomfp = fopen(name,"r");
  if (geomfp==NULL) return UNABLE_TO_OPEN_FILE;

  nvl=fscanf(geomfp,"%lf %5d{n%5d{n",MaxMaturity,MaxExercis
    eDates,Dimension);
  if(nvl!=3) return BAD_TESSELATION_FORMAT;

  TSize=(int*)malloc((*MaxExerciseDates+1)*sizeof(int));

  for (i=0;i<=(*MaxExerciseDates);i++)
    {
      nvl=fscanf(geomfp," %5i",TSize+i);
      if(nvl!=1) return BAD_TESSELATION_FORMAT;
    }
```

```
  nvl=fscanf(geomfp,"%d{n",Nbpt);
  if(nvl!=1) return BAD_TESSELATION_FORMAT;

  if((Grid=(double*)malloc((*Nbpt)*((*Dimension)+1)*sizeof(
    double)))==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  for (id=0;id<*Nbpt;id++)
    {
      nvl=fscanf(geomfp,"%lf ",&t);
      if(nvl!=1) return BAD_TESSELATION_FORMAT;
      for(k=0;k<(*Dimension)+1;k++)
        {
          nvl=fscanf(geomfp,"%lf ",Grid+id*((*Dimension)+1)
    +k);
          if(nvl!=1) return BAD_TESSELATION_FORMAT;
        }
    }
  fclose(geomfp);
  return OK;
}

static int read_graph(int Dim, int Nbpt, int *TSizeMin, cha
    r *name)
{
  int id,j,n;
  FILE *ifp;

  ifp=fopen(name,"rb");
  if (ifp==NULL) return UNABLE_TO_OPEN_FILE;

  fread(&n,sizeof(int),1,ifp);
  if(n!=Nbpt) return BAD_TESSELATION_FORMAT;

  fread(TSizeMin,sizeof(int),1,ifp);

  /*   Voir routine d'allocation */
  Succ=(int*)malloc(Nbpt*(*TSizeMin+1)*sizeof(int));
  Trans=(double*)malloc(Nbpt*(*TSizeMin)*sizeof(double));

  for(id=0;id<Nbpt;id++){
```

```
    /* Lecture du nombre de successeur de id */
    n = fread(Succ+id*(*TSizeMin+1),sizeof(int),1,ifp);
    if (n != 1) break;
    for(j=1;j<=*(Succ+id*(*TSizeMin+1));j++){
      fread(Succ+id*(*TSizeMin+1)+j,sizeof(int),1,ifp);
    }
    for(j=0;j<*(Succ+id*(*TSizeMin+1));j++){
      fread(Trans+id*(*TSizeMin)+j,sizeof(double),1,ifp);
    }
  }
  fclose(ifp);
  return OK;
}

static int MarkovIteration(int id, int Dimension, int TSiz
    eMin, int generator)
{
  double aux, s;
  int k;
  aux=(int)(pnl_rand_uni(generator) * ((int)Grid[id*(Dimens
    ion+1)]))+1;
  s=0.0;
  k=0;
  do {
    s+=Trans[id*TSizeMin+k];
    k++;
  } while (s<aux && k<Succ[id*(TSizeMin+1)]);
  return Succ[id*(TSizeMin+1)+k];
}


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



/* Control variate variable for CallBasket = CallGeom with
```

```
    ad hoc coefficients*/
static double CFControlVariateCallBasket(double time,
    NumFunc_nd *p, PnlVect *VStock,
                                          double BS_Intere
    st_Rate, PnlVect *BS_Dividend_Rate,
                                          PnlVect *_BS_
    Volatility, double *_BS_Correlation)
{
  int i;
  double *Stock = VStock->array;
  int BS_Dimension = VStock->size;
  double aux=exp(log(Stock[0])/BS_Dimension);
  double d1,d2,call;

  if (time==0.){
    return p->Compute(p->Par, VStock);
  }
  for(i=1;i<BS_Dimension;i++){
    aux*=exp(1./BS_Dimension*log(Stock[i]));
  }
  aux=exp(log(aux));
  d2=1./(beta_Basket_BS_Correlation*sqrt(time))*( log(aux/
    p->Par[0].Val.V_DOUBLE) + (BS_Interest_Rate - delta_Basket_
    BS_Dividend_Rate - sigma_Basket_BS_Volatility*sigma_Basket_
    BS_Volatility/2.0)*time);
  d1=d2+beta_Basket_BS_Correlation*sqrt(time);
  call=aux*exp(-delta_Basket_BS_Dividend_Rate*time - sigma_
    Basket_BS_Volatility*sigma_Basket_BS_Volatility*time/2.0 *
    time + beta_Basket_BS_Correlation*beta_Basket_BS_Correlation/2
    .0*time)*cdf_nor(d1);
  call-=p->Par[0].Val.V_DOUBLE*exp(-BS_Interest_Rate*time)*
    cdf_nor(d2);
  return call;
}

/* Control variate variable for PutBasket = PutGeom with ad
     hoc coefficients */
static double CFControlVariatePutBasket(double time,
    NumFunc_nd *p, PnlVect *VStock,
                                         double BS_Interest_
    Rate, PnlVect *BS_Dividend_Rate,
```

```
                                          PnlVect *_BS_
    Volatility, double *_BS_Correlation)
{
  int i;
  int BS_Dimension = VStock->size;
  double *Stock = VStock->array;
  double aux=exp(1./BS_Dimension*log(Stock[0]));


  if(time==0.){
    return p->Compute(p->Par, VStock);
  }
  for(i=1;i<BS_Dimension;i++){
    aux*=exp(1./BS_Dimension*log(Stock[i]));
  }
  aux=exp(log(aux));
  return CFControlVariateCallBasket(time,p,VStock,BS_Intere
    st_Rate,BS_Dividend_Rate,_BS_Volatility, _BS_Correlation) -
    aux*exp(-delta_Basket_BS_Dividend_Rate*time - sigma_Basket_
    BS_Volatility*sigma_Basket_BS_Volatility*time/2.0*time + bet
    a_Basket_BS_Correlation*beta_Basket_BS_Correlation/2.0*
    time) + p->Par[0].Val.V_DOUBLE*exp(-BS_Interest_Rate*time);
}

static double CFdefault(double time, NumFunc_nd *p, PnlVec
    t *VStock,
                        double BS_Interest_Rate,PnlVect *
    BS_Dividend_Rate,
                        PnlVect *_BS_Volatility, double *_
    BS_Correlation)
{
  return 0.;
}

/* Closed formula for CallGeom */
static double CFCallGeom(double time, NumFunc_nd *p, PnlVec
    t *VStock,
                         double BS_Interest_Rate, PnlVect *
    BS_Dividend_Rate,
                         PnlVect *_BS_Volatility, double *_
    BS_Correlation)
```

```
{
  int i;
  int BS_Dimension = VStock->size;
  double *Stock = VStock->array;
  double aux=Stock[0];
  double d1,d2,call,dim;

  dim=(double)BS_Dimension;
  if (time==0.){
    return p->Compute(p->Par, VStock);
  }
  for(i=1;i<BS_Dimension;i++){
    aux*=Stock[i];
  }
  aux=exp(log(aux)/dim);
  d2=dim/(beta_BS_Correlation*sqrt(time))*(log(aux/p->Par[0
    ].Val.V_DOUBLE) + (dim*BS_Interest_Rate - delta_BS_Divid
    end_Rate - sigma_BS_Volatility*sigma_BS_Volatility/2.0)*(
    time/dim));
  d1=d2+beta_BS_Correlation/dim*sqrt(time);
  call=aux*exp(-delta_BS_Dividend_Rate*time/dim - sigma_BS_
    Volatility*sigma_BS_Volatility/(2.0*dim)*time + beta_BS_Correlation*
    beta_BS_Correlation/(2.0*dim*dim)*time)*cdf_nor(d1);
  call-=p->Par[0].Val.V_DOUBLE*exp(-BS_Interest_Rate*time)*
    cdf_nor(d2);
  return call;
}

/* Closed formula for PutGeom */
static double CFPutGeom(double time, NumFunc_nd *p, PnlVec
    t *VStock,
                        double BS_Interest_Rate, PnlVect *
    BS_Dividend_Rate,
                        PnlVect *_BS_Volatility, double *_
    BS_Correlation)
{
  int i;
  double *Stock = VStock->array;
  int BS_Dimension = VStock->size;
  double aux=Stock[0],dim;
```

```
  dim=(double)BS_Dimension;
  if (time==0.){
    return p->Compute(p->Par, VStock);
  }
  for(i=1;i<BS_Dimension;i++){
    aux*=Stock[i];
  }
  aux=exp(log(aux)/dim);
  return CFCallGeom(time, p, VStock, BS_Interest_Rate,BS_
    Dividend_Rate,_BS_Volatility, _BS_Correlation) - aux*exp(-de
    lta_BS_Dividend_Rate*time/dim - sigma_BS_Volatility*sigma_
    BS_Volatility/(2.0*dim)*time + beta_BS_Correlation*beta_BS_
    Correlation/(2.0*dim*dim)*time) + p->Par[0].Val.V_DOUBLE*exp(-BS_
    Interest_Rate*time);
}


/* Closed formula for European PutMin in dimension 2 */
double CFPutMin(double time, NumFunc_nd *p, PnlVect *VStoc
    k,
                double BS_Interest_Rate, PnlVect *BS_Divid
    end_Rate,
                PnlVect *BS_Volatility, double *BS_
    Correlation)
{
  double s1,s2,sigma1,sigma2,rho,divid1,divid2,r,t,k;
  double b1,b2,sigma,rho1,rho2,d,d1,d2,c0,c1;
  double *Stock = VStock->array;
  double price=0.0, delta=0.0;

  if (time==0.) {
    return p->Compute(p->Par,VStock);
  }

  s1=Stock[0];
  s2=Stock[1];
  t=time;
  k=p->Par[0].Val.V_DOUBLE;
  r=BS_Interest_Rate;
  divid1=BS_Dividend_Rate->array[0];
```

```
divid2=BS_Dividend_Rate->array[1];

b1=r-divid1;
b2=r-divid2;

rho=BS_Correlation[1];
sigma1=BS_Volatility->array[0];
sigma2=BS_Volatility->array[1];

sigma=sqrt(SQR(sigma1)+SQR(sigma2)-2*rho*sigma1*sigma2);
if (((sigma-PRECISION)<=0.)&&((rho+PRECISION)>=1.)) {
  if ((s1*exp(-divid1*t))<=(s2*exp(-divid2*t))) {
    pnl_cf_put_bs(s1,k,t,r,divid1,sigma1,&price,&delta);
    return price;
  }
  else
    {
      return pnl_cf_put_bs(s2,k,t,r,divid2,sigma2,&price,
  &delta);
      return price;
    }
}
else
  {
    rho1=(sigma1-rho*sigma2)/sigma;
    rho2=(sigma2-rho*sigma1)/sigma;
    d=(log(s1/s2)+(b1-b2+SQR(sigma)/2.0)*t)/(sigma*sqrt(
  t));
    d1=(log(s1/k)+ (b1+SQR(sigma1)/2.0)*t)/(sigma1*sqrt(
  t));
    d2=(log(s2/k)+ (b2+SQR(sigma2)/2.0)*t)/(sigma2*sqrt(
  t));

    c0=s1*exp((b1-r)*t)*(1.0-cdf_nor(d))+s2*exp((b2-r)*t)
  *cdf_nor(d-sigma*sqrt(t));
    c1=s1*exp((b1-r)*t)*pnl_cdf2nor(d1,-d,-rho1)
      +s2*exp((b2-r)*t)*pnl_cdf2nor(d2,d-sigma*sqrt(t),-
  rho2)
      -k*exp(-r*t)*pnl_cdf2nor(d1-sigma1*sqrt(t),d2-sigma
  2*sqrt(t),rho);
```

```
      /*Price*/
      return k*exp(-r*t)-c0+c1;
    }

}

void CheckParameterOP(NumFunc_nd *p,double _BS_Interest_Ra
    te, PnlVect *_BS_Dividend_Rate,
                      PnlVect *_BS_Volatility, double *_BS_
    Correlation,
                      double (**UneCF)(double time,
    NumFunc_nd *p, PnlVect *VStock,
                                       double BS_Interest_
    Rate, PnlVect *BS_Dividend_Rate,
                                       PnlVect *BS_
    Volatility, double *BS_Correlation))
{
  int i, j;
  /*initialization of the global variables*/
  int BS_Dimension=_BS_Dividend_Rate->size;
  double *BS_Dividend_Rate=_BS_Dividend_Rate->array;
  double *BS_Volatility=_BS_Volatility->array;

  beta_BS_Correlation=0.;
  beta_Basket_BS_Correlation=0.;
  sigma_BS_Volatility=0.;
  sigma_Basket_BS_Volatility=0.;
  delta_BS_Dividend_Rate=0.;
  delta_Basket_BS_Dividend_Rate=0.;
  for(i=0;i<BS_Dimension;i++){
    beta_Basket_BS_Correlation+=1./SQR(BS_Dimension)*BS_
    Volatility[i]*BS_Volatility[i];
    beta_BS_Correlation+=BS_Volatility[i]*BS_Volatility[i];
    sigma_Basket_BS_Volatility+=1./BS_Dimension*BS_
    Volatility[i]*BS_Volatility[i];
    sigma_BS_Volatility+=BS_Volatility[i]*BS_Volatility[i];
    delta_Basket_BS_Dividend_Rate+=1./BS_Dimension*BS_Divid
    end_Rate[i];
    delta_BS_Dividend_Rate+=BS_Dividend_Rate[i];
    for(j=i+1;j<BS_Dimension;j++){
      beta_Basket_BS_Correlation+=2.0*1./SQR(BS_Dimension)*
```

```
    BS_Volatility[i]*BS_Volatility[j]*_BS_Correlation[i*BS_Dim
    ension+j];
      beta_BS_Correlation+=2.0*BS_Volatility[i]*BS_
    Volatility[j]*_BS_Correlation[i*BS_Dimension+j];
    }
  }
  beta_BS_Correlation=sqrt(beta_BS_Correlation);
  sigma_BS_Volatility=sqrt(sigma_BS_Volatility);
  beta_Basket_BS_Correlation=sqrt(beta_Basket_BS_
    Correlation);
  sigma_Basket_BS_Volatility=sqrt(sigma_Basket_BS_
    Volatility);

  if (p->Compute == CallGeom_nd) *UneCF=CFCallGeom;
  else   if (p->Compute == PutGeom_nd) *UneCF=CFPutGeom;
  else   if (p->Compute == CallBasket_nd) *UneCF=CFControlV
    ariateCallBasket;
  else   if (p->Compute == PutBasket_nd) *UneCF=CFControlV
    ariatePutBasket;
  else   if (p->Compute == PutMin_nd && _BS_Dividend_Rate->
    size==2) *UneCF=CFPutMin;
  else *UneCF= CFdefault;
}


/*see the documentation for the parameters meaning*/
static int QOptst(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,
                  char *AL_Geometry_Name,
                  char *AL_Graph_Name,
                  double *AL_FPrice,
                  double *AL_BPrice,
                  PnlVect *Delta)
{
```

```
int i,j,k,time,compt, ret, init_mc;
int TSizeMin,Geometry_Nbpt,OP_Exercise_Dates;
long l,m,id,id1;
double Step,DiscountStep,aux;
double (*CF)(double time, NumFunc_nd *p, PnlVect *VStock,
             double BS_Interest_Rate, PnlVect *BS_Divid
  end_Rate,
             PnlVect *BS_Volatility, double *BS_
  Correlation);
double MaxMaturity,vol;
int MaxExerciseDates;
double *Daux0,*DauxB,*Daux,*Daux2,*Daux3,*Daux4,*Daux5,*
  aus1, *aus2, *aus3, aux1,aux2;
double *AL_Delta=Delta->array;
int BS_Dimension = BS_Spot->size;
PnlVect VStock;
VStock.size=BS_Dimension;

CheckParameterOP(p, BS_Interest_Rate, BS_Dividend_Rate,
                 BS_Volatility, BS_Correlation, &CF);

if((aus1=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
if((aus2=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
if((aus3=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
if((Daux=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
if((Daux0=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
if((Daux2=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
if((Daux4=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
if((Daux5=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
if((Daux3=(double*)malloc(BS_Dimension*BS_Dimension*size
  of(double)))==NULL) return MEMORY_ALLOCATION_FAILURE;
if((DauxB=(double*)malloc(BS_Dimension*sizeof(double)))==
  NULL) return MEMORY_ALLOCATION_FAILURE;
```

```
/* 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;


/* Lecture de la géométrie
            */
/* Renvoie : MaxMaturity, Grid, Geometry_Nbpt, TSize */
ret=read_geom(&MaxMaturity, &MaxExerciseDates, &BS_Dimens
  ion, &Geometry_Nbpt, AL_Geometry_Name);
if(ret!=OK) return ret;

if(OP_Maturity>MaxMaturity)
  {
    Fprintf(TOSCREEN, "#Error : Maturity too large.{n");
    return FAIL;
  }
OP_Exercise_Dates=MaxExerciseDates*(OP_Maturity/MaxMatu
  rity);

/* Lecture du graphe des transitions
            */
/* Renvoie : Succ, Trans, TSizeMin */
ret=read_graph(BS_Dimension, Geometry_Nbpt,&TSizeMin, AL_
  Graph_Name);
if (ret!=OK) return ret;
/*time step*/
Step=MaxMaturity/(double)(MaxExerciseDates);
/*discounting factor for a time step*/
DiscountStep=exp(-BS_Interest_Rate*Step);

/*memory allocation of the BlackScholes variables*/
Init_BS(BS_Dimension,BS_Volatility->array,
        BS_Correlation,BS_Interest_Rate,BS_Dividend_Rate-
  >array);
/*memory allocation of the algorithm's variables*/
ret=QOptStored_Allocation(BS_Dimension,Geometry_Nbpt);
```

```
if (ret!=OK) return ret;
/*initialisation of the dynamical programming prices at
  the maturity*/
time=0;
compt=0;
while(time<=OP_Exercise_Dates-1){
  compt+=TSize[time];
  time+=1;
}
for(id=compt;id<Geometry_Nbpt;id++){
  BlackScholes_Transformation((double)OP_Exercise_Dates*
  Step,BSQ+id*BS_Dimension,Grid+id*(BS_Dimension+1)+1,BS_Dimens
  ion,BS_Spot->array);
  VStock.array=BSQ+id*BS_Dimension;
  Price[id]=p->Compute(p->Par, &VStock) - CF(0.0,p, &VS
  tock,  BS_Interest_Rate, BS_Dividend_Rate,
                                                BS_
  Volatility, BS_Correlation);
}
/*dynamical programming algorithm*/
for (i=OP_Exercise_Dates-1;i>=0;i--){
  compt-=TSize[i];
  /*approximation of the conditionnal expectations*/
  for (j=0;j<TSize[i];j++){
    aux=0;
    id=compt+j;
    for (k=1;k<=Succ[id*(TSizeMin+1)];k++){
      id1=Succ[id*(TSizeMin+1)+k];
      if(Grid[id*(BS_Dimension+1)]>machep){
        aux+=Price[id1]*Trans[id*TSizeMin+k-1]/Grid[id*(
  BS_Dimension+1)];
      }
    }
    /*discounting for a time step*/
    aux*=DiscountStep;
    /*exercise decision*/
    BlackScholes_Transformation((double)i*Step,BSQ+id*BS_
  Dimension,Grid+id*(BS_Dimension+1)+1,BS_Dimension,BS_Spot->
  array);
    VStock.array=BSQ+id*BS_Dimension;
    Price[id]=MAX(p->Compute(p->Par, &VStock) - CF((
```

```
  double)(OP_Exercise_Dates - i)*Step,p, &VStock,  BS_Interest_Ra
  te, BS_Dividend_Rate, BS_Volatility, BS_Correlation),aux);
  }
}
i = 0;
/*approximation of the derivatives (d/dt)^(1/2)=(d/dx)
  at time t=0 */
for(l=0;l<BS_Dimension;l++)
  {
    Daux0[l]=0.;  Daux[l]=0.; Daux2[l]=0.; Daux4[l]=0.;
    Daux5[l]=0.;  DauxB[l]=0.;
    for(m=0;m<BS_Dimension;m++)  Daux3[l*BS_Dimension+m]=
  0.;
  }

VStock.array=BSQ+id*BS_Dimension;
aux2=Price[id]  + CF((double)(OP_Exercise_Dates - i)*Step
  ,p, &VStock,  BS_Interest_Rate, BS_Dividend_Rate, BS_
  Volatility, BS_Correlation);
aux2*=Discount((double)i*Step,BS_Interest_Rate);
vol=0.0;

for(l=0;l<BS_Dimension;l++)
  {
    aus2[l]=BSQ[id*BS_Dimension+l]*exp(BS_Dividend_Rate->
  array[l]*(double)i*Step);
    aus2[l]*=Discount((double)i*Step,BS_Interest_Rate);
    vol+=BS_Volatility->array[l]*BS_Volatility->array[l];
  }
vol/=BS_Dimension;
vol=sqrt(vol);

if (BS_Dimension<=4) {
  for (k=1;k<=Succ[id*(TSizeMin+1)];k++)
    {
      id1=Succ[id*(TSizeMin+1)+k];
      VStock.array=BSQ+id1*BS_Dimension;
      aux1=Price[id1] + CF((double)(OP_Exercise_Dates - (
  i+1))*Step,p, &VStock,  BS_Interest_Rate, BS_Dividend_Rate,
   BS_Volatility, BS_Correlation);
      aux1*=Discount((double)(i+1)*Step,BS_Interest_Rate)
```

```
  ;

      for(l=0;l<BS_Dimension;l++)
        {
          aus1[l]=BSQ[id1*BS_Dimension+l]*exp(BS_Dividend
  _Rate->array[l]*(double)(i+1)*Step);
          aus1[l]*=Discount((double)(i+1)*Step,BS_Intere
  st_Rate);
          Daux[l]+=Trans[id*(TSizeMin)+k-1]*(aus1[l] - au
  s2[l])*(aux1 - aux2);
          Daux2[l]+=Trans[id*(TSizeMin)+k-1]*(aus1[l] -
  aus2[l])*(aus1[l] - aus2[l]);
          for (m=l+1;m<BS_Dimension;m++)
            {
              aus1[m]=BSQ[id1*BS_Dimension+m]*exp(BS_Div
  idend_Rate->array[m]*(double)(i+1)*Step);
              aus1[m]*=Discount((double)(i+1)*Step,BS_
  Interest_Rate);
              Daux3[l*BS_Dimension+m]+=Trans[id*(TSizeM
  in)+k-1]*(aus1[l]-aus2[l])*(aus1[m]-aus2[m]);
            }
        }
    }

  for(l=0;l<BS_Dimension;l++)  AL_Delta[l]=Daux[l]/Daux2[
  l];

}
else
  {
    if (vol<=0.25) {
      /* IPP5 */
      for (k=1;k<=Succ[id*(TSizeMin+1)];k++){
        id1=Succ[id*(TSizeMin+1)+k];
        VStock.array=BSQ+id1*BS_Dimension;
        aux1=Price[id1] + CF((double)(OP_Exercise_Dates -
   (i+1))*Step,p, &VStock,  BS_Interest_Rate, BS_Dividend_Ra
  te, BS_Volatility, BS_Correlation);
        aux1*=Discount((double)(i+1)*Step,BS_Interest_Ra
  te);
```

```
      for(l=0;l<BS_Dimension;l++){
        aus1[l]=BSQ[id1*BS_Dimension+l]*exp(BS_Dividend
_Rate->array[l]*(double)(i+1)*Step);
        aus1[l]*=Discount((double)(i+1)*Step,BS_Intere
st_Rate);
        Daux[l]+=Trans[id*(TSizeMin)+k-1]*(aus1[l] - au
s2[l])*aux1;
        Daux2[l]+=Trans[id*(TSizeMin)+k-1]*(aus1[l] -
aus2[l])*aus1[l];
        for(m=0;m<BS_Dimension;m++){
          aus1[m]=BSQ[id1*BS_Dimension+m]*exp(BS_Divid
end_Rate->array[m]*(double)(i+1)*Step);
          aus1[m]*=Discount((double)(i+1)*Step,BS_
Interest_Rate);
          Daux3[l*BS_Dimension+m]+=Trans[id*(TSizeMin)+
k-1]*(aus1[l]-aus2[l])*(aus1[m]-aus2[m]);
        }
      }
    }

    for(l=0;l<BS_Dimension;l++) AL_Delta[l]=Daux[l]/Da
ux2[l];

  }
  else
    {
      if (vol<=0.35)
        { /* IPP4 */
          for (k=1;k<=Succ[id*(TSizeMin+1)];k++){
            id1=Succ[id*(TSizeMin+1)+k];
            VStock.array=BSQ+id1*BS_Dimension;
            aux1=Price[id1] + CF((double)(OP_Exercise_
Dates - (i+1))*Step,p, &VStock,  BS_Interest_Rate, BS_Divid
end_Rate, BS_Volatility, BS_Correlation);
            aux1*=Discount((double)(i+1)*Step,BS_Intere
st_Rate);

            for(l=0;l<BS_Dimension;l++){
              aus1[l]=BSQ[id1*BS_Dimension+l]*exp(BS_
Dividend_Rate->array[l]*(double)(i+1)*Step);
              aus1[l]*=Discount((double)(i+1)*Step,BS_
```

```
Interest_Rate);
              Daux[l]+=Trans[id*(TSizeMin)+k-1]*(aus1[
l] - aus2[l])*aux1;
              Daux2[l]+=Trans[id*(TSizeMin)+k-1]*(aus1[
l] - aus2[l])*(aus1[l] - aus2[l]);
              for(m=0;m<BS_Dimension;m++){
                aus1[m]=BSQ[id1*BS_Dimension+m]*exp(BS_
Dividend_Rate->array[m]*(double)(i+1)*Step);
                aus1[m]*=Discount((double)(i+1)*Step,
BS_Interest_Rate);
                Daux3[l*BS_Dimension+m]+=Trans[id*(TSiz
eMin)+k-1]*(aus1[l]-aus2[l])*(aus1[m]-aus2[m]);
              }
            }
          }

          for(l=0;l<BS_Dimension;l++){
            AL_Delta[l]=Daux[l]/Daux2[l];
          }
        }
      else
        { /* IPPO */
          for (k=1;k<=Succ[id*(TSizeMin+1)];k++){
            id1=Succ[id*(TSizeMin+1)+k];
            VStock.array=BSQ+id1*BS_Dimension;
            aux1=Price[id1] + CF((double)(OP_Exercise_
Dates - (i+1))*Step,p, &VStock,  BS_Interest_Rate, BS_Divid
end_Rate, BS_Volatility, BS_Correlation);
            aux1*=Discount((double)(i+1)*Step,BS_Intere
st_Rate);

            for(l=0;l<BS_Dimension;l++){
              aus1[l]=BSQ[id1*BS_Dimension+l]*exp(BS_
Dividend_Rate->array[l]*(double)(i+1)*Step);
              aus1[l]*=Discount((double)(i+1)*Step,BS_
Interest_Rate);
              Daux[l]+=Trans[id*(TSizeMin)+k-1]*(aus1[
l] - aus2[l])*(aux1 - aux2);
              Daux2[l]+=Trans[id*(TSizeMin)+k-1]*(aus1[
l] - aus2[l])*(aus1[l] - aus2[l]);
              for(m=0;m<BS_Dimension;m++){
```

```
                aus1[m]=BSQ[id1*BS_Dimension+m]*exp(BS_
  Dividend_Rate->array[m]*(double)(i+1)*Step);
                aus1[m]*=Discount((double)(i+1)*Step,
  BS_Interest_Rate);
                Daux3[l*BS_Dimension+m]+=Trans[id*(TSiz
  eMin)+k-1]*(aus1[l]-aus2[l])*(aus1[m]);
              }
            }
          }

          for(l=0;l<BS_Dimension;l++){
            AL_Delta[l]=Daux[l]/Daux2[l];
          }
        }
    }
  }

*AL_BPrice=Price[0] + CF(OP_Maturity,p, BS_Spot,  BS_
  Interest_Rate, BS_Dividend_Rate, BS_Volatility, BS_Correlation)
  ;

/*  Forward price */
*AL_FPrice = 0.;

if (*AL_BPrice==p->Compute(p->Par, BS_Spot)){
  *AL_FPrice=*AL_BPrice;
} else {
  for (l=0;l<AL_MonteCarlo_Iterations;l++){
    /*spot of the brownian motion*/
    i=0;
    id=0;
    /*optimal stopping for a quantized path*/
    do {
      i++;
      id=MarkovIteration(id,BS_Dimension,TSizeMin,    generator);
      VStock.array=BSQ+id*BS_Dimension;
    } while ((i<=OP_Exercise_Dates) &&
             (p->Compute(p->Par,&VStock)<Price[id] +
              CF((double)(OP_Exercise_Dates - i)*Step,p,
  &VStock,
                 BS_Interest_Rate, BS_Dividend_Rate, BS_
```

```
    Volatility, BS_Correlation) - EPS_CONT));
      /*MonteCarlo formulae for the forward price*/
      VStock.array=BSQ+id*BS_Dimension;
      *AL_FPrice+=Discount((double)i*Step,BS_Interest_Rate)
    *(p->Compute(p->Par,&VStock));
    }
    /*output forward price*/
    *AL_FPrice/=(double)AL_MonteCarlo_Iterations;
  }
  free(Daux0); free(DauxB); free(Daux); free(Daux2);
  free(Daux3); free(Daux4); free(Daux5); free(aus1);
  free(aus2);  free(aus3);
  Close();
  return OK;
}
```

```
int CALC(MC_QuantizationStoredND)(void *Opt, void *Mod,
    PricingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r;
  int i, res;
  double *BS_cor;
  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=QOptst(spot,
             ptOpt->PayOff.Val.V_NUMFUNC_ND,
             ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DA
    TE,
             r, divid, sig,
             BS_cor,
             Met->Par[0].Val.V_LONG,
             Met->Par[1].Val.V_ENUM.value,
             Met->Par[3].Val.V_FILENAME,
             Met->Par[2].Val.V_FILENAME,
             &(Met->Res[0].Val.V_DOUBLE),
             &(Met->Res[1].Val.V_DOUBLE),
             Met->Res[2].Val.V_PNLVECT);
  pnl_vect_free(&divid);
  pnl_vect_free (&spot);
  pnl_vect_free (&sig);
  free(BS_cor);

  return res;
}


static int CHK_OPT(MC_QuantizationStoredND)(void *Opt, voi
    d *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT *opt = (TYPEOPT*)(ptOpt->TypeOpt);
  Model* ptMod=(Model*)Mod;
  TYPEMOD *mod = (TYPEMOD*)(ptMod->TypeModel);
```

```
  if (mod->Size.Val.V_PINT>10)
    return WRONG;
  if ((opt->EuOrAm).Val.V_BOOL!=AMER)
    return WRONG;

  return OK;
}

static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  TYPEOPT *opt = (TYPEOPT*)(Opt->TypeOpt);
  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->Res[2].Val.V_PNLVECT=NULL;
      if ((Met->Par[2].Val.V_FILENAME=malloc(sizeof(char)*
    MAX_PATH_LEN))==NULL)
        return MEMORY_ALLOCATION_FAILURE;
      if ((Met->Par[3].Val.V_FILENAME=malloc(sizeof(char)*
    MAX_PATH_LEN))==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    }
  /* some initialisation */
  if(Met->Res[2].Val.V_PNLVECT==NULL)
    Met->Res[2].Val.V_PNLVECT=pnl_vect_create(opt->Size.Val
    .V_PINT);
  else
    pnl_vect_resize(Met->Res[2].Val.V_PNLVECT,opt->Size.Val
    .V_PINT);

  sprintf(Met->Par[2].Val.V_FILENAME, "%s%smb_tes%sgraphe0_
    %dd_10_MC100000", premia_data_dir, path_sep, path_sep,
    opt->Size.Val.V_PINT);
  sprintf(Met->Par[3].Val.V_FILENAME, "%s%smb_tes%sgrille0_
    %dd_10_MC100000", premia_data_dir, path_sep, path_sep,
    opt->Size.Val.V_PINT);

  return OK;
```

```
}

PricingMethod MET(MC_QuantizationStoredND)=
{
  "MC_Quantization_Stored_nd",
  {{"N iterations",LONG,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"Graph File Name", FILENAME, {100}, FORBID, UNSETABLE},
   {"Grig File Name", FILENAME, {100}, FORBID, UNSETABLE},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_QuantizationStoredND),
  {{"Forward Price",DOUBLE,{100},FORBID},{"Backward Price",
    DOUBLE,{100},FORBID},
   {"Delta",PNLVECT,{1},FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_QuantizationStoredND),
  CHK_mc,
  MET(Init)
};


#undef EPS_CONT
```

References