

[Help](#)

```
#include "bs1d_std.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_Exercise_Dates)
{
    if (Q==NULL)
        Q=(double*)malloc(AL_T_Size*OP_Exercise_Dates*BS_Dimension*sizeof(double));
    if (Q==NULL)
        return MEMORY_ALLOCATION_FAILURE;

    if (Trans==NULL)
        Trans=(double*)malloc(OP_Exercise_Dates*AL_T_Size*AL_T_Size*sizeof(double));
    if (Trans==NULL)
        return MEMORY_ALLOCATION_FAILURE;

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

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

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

```

    if (Aux_Path==NULL)
    return MEMORY_ALLOCATION_FAILURE;

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

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

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

    if (Path_Int==NULL)
    Path_Int=(int*)malloc(OP_Exercice_Dates*sizeo
        f(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_EmBS_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 mc_or_qmc,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_Dimension+j]=Initial_Stock[j];

    for (i=Initial_Time+1;i<Initial_Time+Number_Da
        tes;i++){
        for (j=0;j<BS_Dimension;j++){
            Aux_Stock[j]=Sqrt_Step*Gaussians[mc_or_qmc]
                (1, CREATE, 0, 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+=BS_Dimension-j-1;
            aux-=Step*Aux_BS[j];
        }
    }
}

```

```

        Path[i*BS_Dimension+j]=Path[(i-1)*BS_Dimen
        sion+j]*exp(aux);
    }
}

static double Discount(double Time, double BS_
    Interest_Rate)
{
    return exp(-BS_Interest_Rate*Time);
}

static void Init_Tesselations(long AL_MonteCarlo_
    Iterations, int AL_T_Size,int OP_Exercise_Dates,
    int mc_or_qmc,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_Exercise_Dates*BS_Dim
        ension;

    /* Random Quantizers */
    for (i=0;i<AL_T_Size;i++)
        ForwardPath(Q+i*OP_Exercise_Dates*BS_Dimensi
            on,BS_Spot,0,OP_Exercise_Dates,
                mc_or_qmc,generator,BS_Dimension,Ste
                    p,Sqrt_Step);

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

    for (i=0;i<OP_Exercise_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_Exercise_Da
tes,mc_or_qmc,generator,BS_Dimension,Step,Sqrt_Ste
p);

Vimoins=0;
for (i=1;i<OP_Exercise_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_Siz
e+Vi]+=1;
Vimoins=Vi;
}
}
Weights[0]=AL_MonteCarlo_Iterations-AL_T_Size;
for (i=1;i<OP_Exercise_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_Iteratio
ns,NumFunc_1 *p,int size,int Fermeture,int mc_or_
qmc,int generator,int exercise_date_number,
double *s_vector, double t, double r, double divid,
double sigma,int gj_flag)
{
int i,j,k,BS_Dimension=1,dummy;
long l;
double step,Sqrt_Step,DiscountStep,aux,AL_BPri
ce,AL_FPrice;
AL_FPrice=0.0;
*PrixDir=0.;
step=t/(exercise_date_number-1.);
Sqrt_Step=sqrt(step);

```

```

DiscountStep=exp(-r*step);

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

/*Black-Sholes initialization parameters*/
*Sigma=sigma;
Aux_BS[0]=0.5*SQR(sigma)-r+divid;

/* Cells Weights and Transitions probabilities
*/
Init_Tesselations(MC_Iterations,size,exercise_date_number,mc_or_qmc,generator,BS_Dimension,s_vector,step,Sqrt_Step);

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

/* Dynamical programming (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*exercise_date_number*BS_Dimension+(i+1)*BS_Dimension)))*
        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));
    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*
    exercise_date_number*BS_Dimension+
        BS_Dimension))
    )*Trans[size*size+k];
/*Backward Price*/
aux*=DiscountStep;
if(!gj_flag)
AL_BPrice=MAX((p->Compute)(p->Par,s_vector[0]
    ),aux);
else AL_BPrice=aux;

/* Forward price */
for (k=0;k<size;k++){
Price[k]=AL_BPrice-(p->Compute)(p->Par,s_vect
    or[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_da
        te_number*BS_Dimension+i*BS_Dimension)));
}

for (l=0;l<MC_Iterations-size;l++){
ForwardPath(Aux_Path,s_vector,0,exercise_date_
    number,mc_or_qmc,generator,BS_Dimension,step,Sqrt_
        Step);
Path_Int[0]=0;
for (i=1;i<exercise_date_number;i++){
    Path_Int[i]=NearestCell(i,size,exercise_da
        te_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]*exercise_date_number*BS_Dimension+i*BS_Dimen
        sion)));
    }
    AL_FPrice/=(double)MC_Iterations;

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

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

    return;
}

static int MCRandomQuantization(double s,
    NumFunc_1 *p, double t, double r, double divid,
    double sigma, long N, int generator, double inc,int
    exercise_date_number,int size_tesselation,double *
    ptprice, double *ptdelta)
{

    double p1,p2,p3;
    int simulation_dim=1,fermeture=1,init_mc, mc_or
        _qmc;
    double s_vector[1];
    double s_vector_plus[1];

    /*Initialisation*/
    s_vector[0]=s;
    s_vector_plus[0]=s*(1.+inc);

```

```

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

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

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

/*Delta*/
if (exercise_date_number==0){
RaQ(&p1,N,p,size_tesselation,fermeture,mc_or
_qmc,generator,2,s_vector_plus,t,r,divid,sigma,1
);
RaQ(&p2,N,p,size_tesselation,fermeture,mc_or
_qmc,generator,3,s_vector_plus,t,r,divid,sigma,1
);
RaQ(&p3,N,p,size_tesselation,fermeture,mc_or
_qmc,generator,4,s_vector_plus,t,r,divid,sigma,1
);
*ptdelta=((p3+7./2.*(p3-p2)-(p2-p1)/2.)*pt
price)/(s*inc);
} else {
RaQ(&p1,N,p,size_tesselation,fermeture,mc_or
_qmc,generator,exercise_date_number,s_vector_plu

```

```

        s,t,r,divid,sigma,0);
        *ptdelta=(p1-*ptprice)/(s*inc);
    }
}
return init_mc;
}

int CALC(MC_RandomQuantization)(void *Opt, void *
    Mod, PricingMethod *Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    double r,divid;

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

    return MCRandomQuantization(ptMod->S0.Val.V_PD
        OUBLE,
            ptOpt->PayOff.Val.V_
        NUMFUNC_1,
            ptOpt->Maturity.Val.V_DA
        TE-ptMod->T.Val.V_DATE,
            r,
            divid,
            ptMod->Sigma.Val.V_PDOUBL
        E,
            Met->Par[0].Val.V_LONG,
            Met->Par[1].Val.V_INT,
            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));
}

int CHK_OPT(MC_RandomQuantization)(void *Opt, vo
    id *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)
{
    static int first=1;
    int type_generator;

    type_generator= Met->Par[1].Val.V_INT;

    if (first)
    {
        Met->Par[0].Val.V_LONG=20000;
        Met->Par[1].Val.V_INT=0;
        Met->Par[2].Val.V_PDOUBLE=0.1;
        Met->Par[3].Val.V_INT=15;
        Met->Par[4].Val.V_INT=150;
        first=0;
    }
    return OK;
}

PricingMethod MET(MC_RandomQuantization)=
{
    "MC_RandomQuantization",
    {"N iterations",LONG,100,ALLOW},
    {"RandomGenerator",GENER,100,ALLOW},
    {"Delta Increment Rel",PDOUBLE,100,ALLOW},
    {"Number of Exercise Dates (0->Geske Johnson
        Formulae",INT,100,ALLOW},
    {"Tesselation Size",INT,100,ALLOW},
    {" ",END,0,FORBID}},
    CALC(MC_RandomQuantization),

```

```
{{"Price",DOUBLE,100,FORBID},
 {"Delta",DOUBLE,100,FORBID} ,
 {" ",END,0,FORBID}},
CHK_OPT(MC_RandomQuantization),
CHK_mc,
MET(Init)
};
```

References