

[Help](#)

```
#include "bs1d_std.h"

static int d=1;
static long N_sim;
static double **X,**W,**Dw,**ln,**Z,*P,*Pn,*P2,*
    Delta,*Qn,*Semi,*Obst;
static double *drift,*diff_z;
static double *s,r,**sigma,t,*divid,K;

static void memory_allocation()
{
    int i;

    sigma=(double **)calloc(d,sizeof(double *));
    for (i=0;i<d;i++)
        sigma[i]=(double *)calloc(d,sizeof(double));

    s=(double *)malloc((d)*sizeof(double));
    divid=(double *)malloc((d)*sizeof(double));
    drift=(double *)malloc((d)*sizeof(double));
    diff_z=(double *)malloc((d)*sizeof(double));

    X=(double **)calloc(d,sizeof(double *));
    for (i=0;i<d;i++)
        X[i]=(double *)calloc(N_sim,sizeof(double));

    W=(double **)calloc(d,sizeof(double *));
    for (i=0;i<d;i++)
        W[i]=(double *)calloc(N_sim,sizeof(double));

    Dw=(double **)calloc(d,sizeof(double *));
    for (i=0;i<d;i++)
        Dw[i]=(double *)calloc(N_sim,sizeof(double));

    ln=(double **)calloc(d,sizeof(double *));
    for (i=0;i<d;i++)
        ln[i]=(double *)calloc(N_sim,sizeof(double));

    Z=(double **)calloc(d,sizeof(double *));
    for (i=0;i<d;i++)
```

```

    Z[i]=(double *)calloc(N_sim,sizeof(double));

    Pn=(double *)malloc((N_sim)*sizeof(double));
    Qn=(double *)malloc((N_sim)*sizeof(double));
    P=(double *)malloc((N_sim)*sizeof(double));
    P2=(double *)malloc((N_sim)*sizeof(double));
    Delta=(double *)malloc((N_sim)*sizeof(double));
    Semi=(double *)malloc((N_sim)*sizeof(double));
    Obst=(double *)malloc((N_sim)*sizeof(double));

    return;
}

/*Memory Desallocation*/
static void free_memory()
{
    int i;

    for (i=0;i<d;i++)
        free(sigma[i]);
    free(sigma);

    for (i=0;i<d;i++)
        free(X[i]);
    free(X);

    for (i=0;i<d;i++)
        free(W[i]);
    free(W);

    for (i=0;i<d;i++)
        free(Z[i]);
    free(Z);

    for (i=0;i<d;i++)
        free(Dw[i]);
    free(Dw);

    for (i=0;i<d;i++)
        free(ln[i]);
    free(ln);

```

```
    free(divid);
    free(drift);
    free(s);
    free(diff_z);

    free(Pn);
    free(Qn);
    free(P);
    free(P2);
    free(Delta);
    free(Semi);

    return;
}

static double H(double x)
{
    double val;

    if (x>=0.) val=1.;
    else val=0.;

    return val;
}

static double g1(double x,double lambda)
{
    double val;

    val=0.5*lambda*exp(-lambda*dabs(x));

    return val;
}

static double GH1(double x,double lambda)
{
    double val;

    if (x<0.) val=0.5*exp(lambda*x);
    else val=1-0.5*exp(-lambda*x);
```

```

    return val;
}

static int MCLionsRegnier(double x, NumFunc_1 *
    p, double t, double r, double dividp, double sig
    map, long N, int generator, int exercise_date_numbe
    er, double *ptprice, double *ptdelta)
{
    int simulation_dim= 1,fermeture=1,init_mc,
        mc_or_qmc,dummy;
    int i,j,k,jz,TimeIndex,n;
    double eps,sum,sum1,sum2,eps_sqrt,eps2,att,semi
        0;
    double val,tmp1,tmp2;
    double lambda;
    double put_price,put_delta,K;
    double prod1,prod2,prodT,prodT1,prodR,prodR1,su
        mT,sumT1,sumR,sumR1, lambdaT,lambdaT1,lambdaR,la
        mbdaR1,delta;

    N_sim=N;
    n=exercise_date_number;
    memory_allocation();
    K=p->Par[0].Val.V_DOUBLE;
    /*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);
        memory_allocation();
        eps=t/(double)n;
        eps_sqrt=eps*sqrt(eps);
        eps2=SQR(eps);
        att=exp(-r*eps);

```

```

        for (i=0;i<d;i++)
for (j=0;j<=i;j++)
    sigma[i][j]=sigmap;

    /*Drift,Diffusion*/
    for (i=0;i<d;i++)
{
    s[i]=x;
    divid[i]=dividp;
    sum1=0.;
    sum2=0.;

    for (j=0;j<=i;j++)
    {
        sum1+=SQR(sigma[i][j]);
        sum2+=sigma[i][j];
    }
    drift[i]=(r-divid[i]-0.5*sum1)*eps;
    diff_z[i]=sqrt(eps)*sigma[i][i];
}

    /*Brownian motion at the end*/
    for (i=0;i<d;i++)
for (j=0;j<N;j++)

    W[i][j]=Gaussians[mc_or_qmc](1, CREATE, 0, g
    enerator)*sqrt(t);

    /*Final Stock*/
    for (i=0;i<d;i++)
{
    for (j=0;j<N;j++)
    {
        sum=0.;
        for (k=0;k<=i;k++)
        {
            sum+=sigma[i][k]*W[k][j];
        }
        X[i][j]=s[i]*exp(drift[i]*(double)n+sum)
    ;
    }
}

```

```

}

/*Final Price*/
for (j=0;j<N;j++)
Pn[j]=0.0;

/*Backward Cycle*/
for (TimeIndex=n-1;TimeIndex>0;TimeIndex--)
{
tmp1=(double)(TimeIndex)/(double)(TimeIndex+
1);
tmp2=sqrt(tmp1*eps);

/*X,ln,Z,DW*/
for (i=0;i<d;i++)
{
for (j=0;j<N;j++)
{
sum=0.;
val=W[i][j];
W[i][j]=W[i][j]*tmp1+tmp2*Gaussians[mc_or
_qmc](1, CREATE, 0, generator);
for (k=0;k<=i;k++)
sum+=sigma[i][k]*W[k][j];

/*X*/
X[i][j]=s[i]*exp(drift[i]*(double)TimeInd
ex+sum);

Z[i][j]=X[i][j];

Dw[i][j]=eps*W[i][j]-(val-W[i][j])*((
double)TimeIndex*eps)
+eps2*(double)TimeIndex*sigma[i][i];
}
}

/*P,Semi*/
for (j=0;j<N;j++)
{
dummy=Put_BlackScholes_73(X[0][j],K,t-(

```

```

double)TimeIndex*eps,r,divid[0],sigma[0][0],&put_price,
&put_delta);
    Obst[j]=(p->Compute)(p->Par,X[0][j])-
put_price;
    lambda=1./sqrt(eps*(double)TimeIndex);

    sum1=0.;
    sum2=0.;
    for(jz=0;jz<N;jz++)
    {
        prod1=g1(Z[0][jz]-Z[0][j],lambda)+(H(Z[0]
[jz]-Z[0][j])-GH1(Z[0][jz]-Z[0][j],lambda))*(Dw[
0][jz]/Z[0][jz))*(1./(sigma[0][0]*eps2*(double)
TimeIndex));
        prod2=g1(Z[0][jz]-Z[0][j],lambda)+(H(Z[0]
[jz]-Z[0][j])-GH1(Z[0][jz]-Z[0][j],lambda))*(Dw[
0][jz]/Z[0][jz))*(1./(sigma[0][0]*eps2*(double)
TimeIndex));
        sum1+=prod1*Pn[jz];
        sum2+=prod2;
    }

    Semi[j]=sum1/sum2;

    /*Options Values*/
    P[j]=MAX(Obst[j],att*Semi[j]);

    if(TimeIndex==2)
    {
        P2[j]=P[j];
    }

    if(TimeIndex==1)
    {
        if( P[j]==Obst[j])
        {
            Delta[j]=-H(K-Z[0][j])-put_delta;
        }
        else
        {

```

```

        lambdaT1=1./sqrt(eps*(double)TimeInd
ex);
        lambdaT=1./sqrt(eps*(double)TimeInd
ex);
        lambdaR1=1./sqrt(eps*(double)TimeInd
ex);
        lambdaR=1./sqrt(eps*(double)TimeInd
ex);
        sumT=0.;
        sumT1=0.;
        sumR=0.;
        sumR1=0.;

        for(jz=0;jz<N;jz++)
        {
            prodT=g1(Z[0][jz]-Z[0][j],lambdaT)+(H(
Z[0][jz]-Z[0][j])-GH1(Z[0][jz]-Z[0][j],lambdaT))
*(Dw[0][jz]/Z[0][jz))*(1./(sigma[0][0]*eps2*(
double)TimeIndex));

            prodT1=g1(Z[0][jz]-Z[0][j],lambdaT1)+(
H(Z[0][jz]-Z[0][j])-GH1(Z[0][jz]-Z[0][j],lambdaT1
))*(Dw[0][jz]/Z[0][jz))*(1./(sigma[0][0]*eps2*(
double)TimeIndex));

            prodR=-g1(Z[0][jz]-Z[0][j],lambdaR)*(
Dw[0][jz]/Z[0][jz))*(1./(sigma[0][0]*eps2*(
double)TimeIndex))-(H(Z[0][jz]-Z[0][j])-GH1(Z[0][jz]-Z
[0][j],lambdaR))*(1./(sigma[0][0]*eps2*(double)
TimeIndex))*(1./SQR(Z[0][jz]))*(SQR(Dw[0][jz]))*(1./(si
gma[0][0]*eps2*(double)TimeIndex))+Dw[0][jz]-((
eps*(double)TimeIndex)/sigma[0][0]));

            prodR1=-g1(Z[0][jz]-Z[0][j],lambdaR1)*
(Dw[0][jz]/Z[0][jz))*(1./(sigma[0][0]*eps2*(
double)TimeIndex))-(H(Z[0][jz]-Z[0][j])-GH1(Z[0][jz]-Z
[0][j],lambdaR1))*(1./(sigma[0][0]*eps2*(double)
TimeIndex))*(1./SQR(Z[0][jz]))*(SQR(Dw[0][jz]))*(1./(si
gma[0][0]*eps2*(double)TimeIndex))+Dw[0][jz]-((
eps*(double)TimeIndex)/sigma[0][0]));

```

```

        sumT+=prodT*P2[jz];
        sumT1+=prodT1;
        sumR+=prodR*P2[jz];
        sumR1+=prodR1;

    }
    Delta[j]=att*(sumR*sumT1-sumT*sumR1)/
(SQR(sumT1));
    }
}

for (j=0;j<N;j++)
    Pn[j]=P[j];
}

/*Final Step*/
dummy=Put_BlackScholes_73(x,K,t,r,divid[0],
sigma[0][0],&put_price,&put_delta);

sum=0.;
for (jz=0;jz<N;jz++)
sum+=Pn[jz];

semi0=sum/(double)N;

sum=0.;
for (jz=0;jz<N;jz++)
sum+=Delta[jz];

delta=sum/(double)N+put_delta;

*ptprice=MAX((p->Compute)(p->Par,s[0])-put_
price,att*semi0)+put_price;
*ptdelta=delta;
}

free_memory();
return init_mc;
}

```

```

int CALC(MC_LionsRegnier)(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 MCLionsRegnier(ptMod->S0.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_1,
        ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.
        V_DATE,
        r,
        divid,
        ptMod->Sigma.Val.V_PDOUBLE,
        Met->Par[0].Val.V_LONG,
        Met->Par[1].Val.V_INT,
        Met->Par[2].Val.V_INT,
        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE));
}

int CHK_OPT(MC_LionsRegnier)(void *Opt, void *Mod
    )
{
    Option* ptOpt=(Option*)Opt;
    TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

    if ((strcmp( ((Option*)Opt)->Name,"PutAmer")==0
        ) )
        return OK;
    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=500;
    Met->Par[1].Val.V_INT=0;
    Met->Par[2].Val.V_INT=20;
    first=0;
}
return OK;
}

PricingMethod MET(MC_LionsRegnier)=
{
    "MC_LionsRegnier",
    {{ "N iterations", LONG, 100, ALLOW},
      {"RandomGenerator", GENER, 100, ALLOW},
      {"Number of Exercise Dates", INT, 100, ALLOW},
      {" ", END, 0, FORBID}}},
    CALC(MC_LionsRegnier),
    {{ "Price", DOUBLE, 100, FORBID},
      {"Delta", DOUBLE, 100, FORBID} ,
      {" ", END, 0, FORBID}}},
    CHK_OPT(MC_LionsRegnier),
    CHK_mc,
    MET(Init)
};

```

References