Assignment 7

Chao Cheng

April 2, 2019

1 Question 1

Implement the recursive random walk algorithm and the Brownian bridge algorithm to simulate a standard one-dimensional Brownian motion. Check the validity of your results by calculating appropriate statistics.

1.1 Recursive Brownian Walk

1.1.1 code

```
#include<iostream>
#include<cmath>
#define pis 8.0*atan(1)
using namespace std;

double boxMuller(){
    double u1,u2,z;     //u(0,1),gsn(0,1)
    u1 = ((double)rand()/(RAND_MAX));
    u2 = ((double)rand()/(RAND_MAX));
    z = sqrt(-2*log(u1))*cos(pis*u2);
    return z;
}

int main(){
    double z;         //gsn(0,1)
    int count = 8;  //total walk steps
    double w[count+1]={0}; //positions
    double increase;
    double times[count+1] = {0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8};//time specified
    int N = 100000; //total number of Brownian Motion generated
    double mean[count]={0.0},mean2[count]={0.0},variance[count]={0.0};

    srand(1) ;  // set up random seed
    w[0] = 0.0; //initialize position
    for (int j=0; j<N; j++) {

        for(int i = 1; i <= count; i++)
        {
            z = boxMuller();
            increase = sqrt(times[i]-times[i-1])*z;
            w[i] = w[i-1] + increase;
            mean[i-1] = mean[i-1] + increase;
            mean2[i-1] = mean2[i-1] + increase*increase;
        }
    }
    for (int i = 0; i < count; i++) {
        variance[i] = (mean2[i] - mean[i]*mean[i]/N)/(N-1);
        mean[i] = mean[i]/N;
    }
    for(int i=0;i<count;i++){
    cout<<"The mean of increment of Brownian Motion is "<<mean[i]<<endl;
    cout<<"The variance of increment of Brownian Motion is "<<variance[i]<<endl;
    cout<<endl;
    }
}
```

1.1.2 Results

```
The mean of increment of Brownian Motion is -0.000919543
The variance of increment of Brownian Motion is 0.100545

The mean of increment of Brownian Motion is 0.000242216
The variance of increment of Brownian Motion is 0.0998289

The mean of increment of Brownian Motion is -0.00051573
```

```
The variance of increment of Brownian Motion is 0.100464

The mean of increment of Brownian Motion is 0.000949916
The variance of increment of Brownian Motion is 0.0997419

The mean of increment of Brownian Motion is 0.00124292
The variance of increment of Brownian Motion is 0.10031

The mean of increment of Brownian Motion is -0.0015778
The variance of increment of Brownian Motion is 0.0998082

The mean of increment of Brownian Motion is -0.0012623
The variance of increment of Brownian Motion is 0.0993423

The mean of increment of Brownian Motion is 0.00146799
The variance of increment of Brownian Motion is 0.0996125
```

1.2 Brownian Bridge Algorithm

1.2.1 code

```
#include<iostream>
#include<cmath>
#define pis 8.0*atan(1)
using namespace std;

double boxMuller(){
    double u1,u2,z;     //u(0,1),gsn(0,1)
    double w;
    u1 = ((double)rand()/(RAND_MAX));
    u2 = ((double)rand()/(RAND_MAX));
    z = sqrt(-2*log(u1))*cos(pis*u2);
    return z;
}
void swap(double *xp, double *yp)
{
    double temp = *xp;
    *xp = *yp;
    *yp = temp;
}
// A function to implement bubble sort
void bubbleSort(double arr[], double arr2[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (arr[j] > arr[j+1]) {
               swap(&arr[j], &arr[j+1]);
               swap(&arr2[j], &arr2[j+1]);
           }

}
int main(){
    double z;        //gsn(0,1)
    double wbegin = 0.0,tbegin = 0.0;
    double wend,tend = 0.8;
    double diffT, increase;
    int count = 4;
    double w[count+1] = {0,0}; //positions
    double t[count+1] = {0.0};//time
    double coVariance[count][count];//covariance of brownian positions
    double mean[count]={0},mean2[count]={0},variance[count]={0};
    int N = 100000,j; //total number of Brownian Motion generated

    srand(1) ;  // set up random seed
    for (int i=0; i<N; i++) {
        z = boxMuller();
        wend = wbegin + sqrt(tend-tbegin)*z; //end of the bridge
        w[0] = wbegin;
        w[1] = wend;
        t[0] = tbegin;
        t[1] = tend;
        diffT = tend - tbegin;
        j = 2;
```

```
        for(int j =2;j<count+1;j++){
            z = boxMuller();
            w[j] = 0.5*(w[j-1] + w[j-2]) + 0.5 * sqrt(diffT) * z;
            t[j] = 0.5*(t[j-1] + t[j-2]);
            diffT = abs(t[j]-t[j-1]);
        }
//================bubble sort====================
         bubbleSort(t, w , count+1);
//================================================
        for (int k = 0; k<count; k++) {
            increase = w[k+1] - w[k];
            mean[k] = mean[k] + increase;
            mean2[k] = mean2[k] + increase*increase;
        }

    }
    for (int i = 0; i < count; i++) {
        variance[i] = (mean2[i] - mean[i]*mean[i]/N)/(N-1);
        mean[i] = mean[i]/N;
    }
    for(int i=0;i<count;i++){
    cout<<"The mean of increment of Brownian Motion is "<<mean[i]<<endl;
    cout<<"The variance of increment of Brownian Motion is "<<variance[i]<<endl;
    cout<<endl;
    }
}
```

1.2.2 Results

```
The mean of increment of Brownian Motion is -0.000584307
The variance of increment of Brownian Motion is 0.400148

The mean of increment of Brownian Motion is 0.0013439
The variance of increment of Brownian Motion is 0.100518

The mean of increment of Brownian Motion is 0.000142499
The variance of increment of Brownian Motion is 0.0997896

The mean of increment of Brownian Motion is 0.00212676
The variance of increment of Brownian Motion is 0.200228
```

2 Question 2

In Homework 5, you were asked to use Monte Carlo integration to approximate

$$\int_0^1 e^{x^2} dx$$

Use antithetic variates in a simulation to approximate this integral when using $N = 10^i$ realizations for $i = 2, \cdots, 6$. Compare the variance of your estimator when using antithetic variates with the variance of your previous results obtained without the use of a variance reduction technique.

2.1 Code

```
#include<iostream>
#include<cmath>
#include <tuple>
#define pis 8.0*atan(1)
using namespace std;

tuple<double,double,double,double> MCIntegrelVar(int n){
    double u;           // u(0,1),
    double x, y;           //realization of exp(u^2)
    double est,estA;         //est.(integral), antithetic est.
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of antithetic

    u = ((double)rand()/(RAND_MAX));
    x = exp(u*u);
    y = exp((1-u)*(1-u));
    est = x ;
    estA = (x+y)/2;
    svs = 0;
    svsA = 0;
    mean = est;
    meanA = estA;
    for(int i = 1; i < n; i++)
    {
         u = ((double)rand()/(RAND_MAX));
         x = exp(u*u);
         y = exp((1-u)*(1-u));
         est = est + (x - est)/(i+1);
         estA = estA + ((x+y)/2-estA)/(i+1);
         svs = svs *(i - 1)/i  + (mean - est)*(mean - est)/(i + 1);
         svsA = svsA *(i - 1)/i  + (meanA - estA)*(meanA - estA)/(i + 1);
         mean = mean + (est - mean)/(i+1);
         meanA = meanA + (estA - meanA)/(i+1);
    }
    return make_tuple(mean, meanA, svs,svsA);
}
int main(){
    int N[5]={100,1000,10000,100000,1000000};
    for(int i=0;i<5;i++){
        srand(1);       //set up the same random seed for comparison
        auto Var = MCIntegrelVar(N[i]);
        cout<<"For N = "<<N[i]<<endl;
        cout<<"Value of the MC integrel is "<<get<0>(Var)<<endl;
        cout<<"Value of antithetic MC Integrel is "<<get<1>(Var)<<endl;
        cout<<"Variance of normal MC Integrel is "<<get<2>(Var)<<endl;
        cout<<"Variance of antithetic MC Integrel is "<<get<3>(Var)<<endl;
        cout<<endl;
    }
}
```

2.2 Results

```
For N = 100
Value of the MC integrel is 1.52395
Value of antithetic MC Integrel is 1.45482
Variance of normal MC Integrel is 0.00599921
Variance of antithetic MC Integrel is 0.000331599

For N = 1000
Value of the MC integrel is 1.48865
Value of antithetic MC Integrel is 1.45837
Variance of normal MC Integrel is 0.000991057
```

```
Variance of antithetic MC Integrel is 6.6208e-05

For N = 10000
Value of the MC integrel is 1.46457
Value of antithetic MC Integrel is 1.4619
Variance of normal MC Integrel is 0.000167223
Variance of antithetic MC Integrel is 9.80793e-06

For N = 100000
Value of the MC integrel is 1.46105
Value of antithetic MC Integrel is 1.46245
Variance of normal MC Integrel is 2.02715e-05
Variance of antithetic MC Integrel is 1.06768e-06

For N = 1000000
Value of the MC integrel is 1.46238
Value of antithetic MC Integrel is 1.4624
Variance of normal MC Integrel is 2.28581e-06
Variance of antithetic MC Integrel is 1.24903e-07
```

3 Question 3

In Homeworks 1 and 5, you were asked to estimate $E[M]$ where M is equal to the number of uniformly distributed on (0,1) random number that must be summed to exceed 1. In other words, for uniformly distributed on (0,1) random variables $U_1, U_2, \cdots$,

$$M = \min\{n : \sum_{i=1}^{n} U_i > 1\}.$$

You observed that it appeared that the expected value was e. Use antithetic variates in a simulation to estimate e when using $N = 10^i$ realizations for $i = 2, \cdots, 6$. Compare the variance of your estimator when using antithetic variates with the variance of your previous results obtained without the use of variance reduction technique.

3.1 Code

```
#include<iostream>
#include<cmath>
#include <tuple>
using namespace std;
tuple<double,double> numGreatOne(){
    double u,sum,sumA;//u~(0,1), sum of us, sum of antithetics
    double i,j;
    sum = 0;
    sumA = 0;
    i = 0;
    j = 0;
    while( (sum<= 1) || (sumA <= 1) ){
        u = ((double)rand()/(RAND_MAX));
        if(sum<= 1){
            sum = sum + u;
            i = i + 1;
        }
        if(sumA<=1){
            sumA = sumA + 1 - u;
            j = j + 1;
        }
    }
    return make_tuple(i,j);
}
tuple<double,double,double,double> getVars(int n){
    double u;           // u(0,1),
    double x, y, xA;    // realizations
    double mean, svs;   //mean, sample variance of estimator
    double meanA,svsA; //mean, var of antithetic

    auto results = numGreatOne();
    x = get<0>(results);
    y = get<1>(results);
    xA = (x+y)/2;
    mean = x;
    meanA = xA;
    svs = 0;
    svsA = 0;
    for(int i = 1; i < n; i++)
    {
        auto results = numGreatOne();
        x = double(get<0>(results));
        y = double(get<1>(results));
        xA = (x+y)/2;
        svs = svs *(i - 1)/i  + (mean - x)*(mean - x)/(i+1);
        mean = mean + (x-mean)/(i+1);

        svsA = svsA *(i - 1)/i  + (meanA - xA)*(meanA - xA)/(i+1);
        meanA = meanA + (xA-meanA)/(i+1);
    }
    return make_tuple(mean, meanA, svs,svsA);
}

int main(){
    int N[5]={100,1000,10000,100000,1000000};
    for(int i=0;i<5;i++){
        srand(1);       //set up the same random seed for comparison
        auto Var = getVars(N[i]);
        cout<<"For N= "<<N[i]<<endl;
        cout<<"Value of the M is "<<get<0>(Var)<<endl;
        cout<<"Value of antithetic M is "<<get<1>(Var)<<endl;
```

```
        cout<<"Variance of normal M is "<<get<2>(Var)<<endl;
        cout<<"Variance of antithetic M is "<<get<3>(Var)<<endl;
        cout<<endl;
    }
}
```

3.2 Results

```
For N = 100
Value of the M is 2.69
Value of antithetic M is 2.77
Variance of normal M is 0.822121
Variance of antithetic M is 0.153636

For N = 1000
Value of the M is 2.716
Value of antithetic M is 2.714
Variance of normal M is 0.78613
Variance of antithetic M is 0.127331

For N = 10000
Value of the M is 2.7157
Value of antithetic M is 2.7164
Variance of normal M is 0.76895
Variance of antithetic M is 0.126284

For N = 100000
Value of the M is 2.70965
Value of antithetic M is 2.71452
Variance of normal M is 0.750474
Variance of antithetic M is 0.122797

For N = 1000000
Value of the M is 2.71717
Value of antithetic M is 2.71763
Variance of normal M is 0.76412
Variance of antithetic M is 0.1245
```

4 Question 4

In Homework 5, you were asked to estimate $E(L)$ where L denotes the first uniformly distributed on (0,1) random number that is less than its predecessor. In other words, for uniformly distributed on (0,1) random variables $U_1, U_2, \cdots$,

$$L = \min\{n : U_1 \leq U_2 \leq \cdots \leq U_{n-1} > U_n\}.$$

It can be shown that $E[L] = e$. Use antithetic variates in a simulation to estimate $E[L]$ using $N = 10^i$ realizations for $i = 2, \cdots, 6$. Compare the variance of your estimator when using antithetic variates with the variance of your previous results obtained without the use of a variance reduction technique.

4.1 Code

```
#include<iostream>
#include<cmath>
#include <tuple>
using namespace std;
tuple<double,double> numTrail(){
    double u,uA,newU,newUA;//u~(0,1)
    double tmp,uold,newUold;
    double i,j;
    u = 0;
    uA = 0;
    newU = ((double)rand()/(RAND_MAX));
    newUA = 1 - newU;
    i = 1;
    j = 1;

    while((u <= newU) || (uA <= newUA)){
        uold = u;
        newUold = newU;
        if (u <= newU) {
            u = newU;
            newU = ((double)rand()/(RAND_MAX));
            i = i + 1 ;
        }

        if (uA <= newUA) {
             uA = newUA;
             if(uold <= newUold){
                 newUA = 1 - newU;
             }else{
                 newUA = 1 - ((double)rand()/(RAND_MAX));
             }
             j = j + 1 ;
         }
    }
    return make_tuple(i,j);
}
tuple<double,double,double,double> getVars(int n){
    double x, y, xA;   // realizations
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of antithetic

    auto results = numTrail();
    x = get<0>(results);
    y = get<1>(results);
    xA = (x+y)/2;
    mean = x;
    meanA = xA;
    svs = 0;
    svsA = 0;
    for(int i = 1; i < n; i++)
    {
        auto results = numTrail();
        x = double(get<0>(results));
        y = double(get<1>(results));
        xA = (x+y)/2;
        svs = svs *(i - 1)/i  + (mean - x)*(mean - x)/(i+1);
        mean = mean + (x-mean)/(i+1);

        svsA = svsA *(i - 1)/i  + (meanA - xA)*(meanA - xA)/(i+1);
        meanA = meanA + (xA-meanA)/(i+1);
    }
    return make_tuple(mean, meanA, svs, svsA);
```

```
}

int main(){
    int N[5]={100,1000,10000,100000,1000000};
    for(int i=0;i<5;i++){
        srand(1);       //set up the same random seed for comparison
        auto Var = getVars(N[i]);
        cout<<"For N = "<<N[i]<<endl;
        cout<<"Value of the L is "<<get<0>(Var)<<endl;
        cout<<"Value of antithetic L is "<<get<1>(Var)<<endl;
        cout<<"Variance of normal L is "<<get<2>(Var)<<endl;
        cout<<"Variance of antithetic L is "<<get<3>(Var)<<endl;
        cout<<endl;
    }
}
```

4.2 Results

```
For N = 100
Value of the L is 2.68
Value of antithetic L is 2.735
Variance of normal L is 0.825859
Variance of antithetic L is 0.153813

For N = 1000
Value of the L is 2.68
Value of antithetic L is 2.721
Variance of normal L is 0.744344
Variance of antithetic L is 0.136295

For N = 10000
Value of the L is 2.7135
Value of antithetic L is 2.7218
Variance of normal L is 0.778296
Variance of antithetic L is 0.129868

For N = 100000
Value of the L is 2.71634
Value of antithetic L is 2.7162
Variance of normal L is 0.766405
Variance of antithetic L is 0.123904

For N = 1000000
Value of the L is 2.71655
Value of antithetic L is 2.71807
Variance of normal L is 0.7635
Variance of antithetic L is 0.124543
```

5 Question 5

In Homework 3 and 5, you were asked to continually roll a pair of fair dice until all possible outcomes $2, 3, \cdots, 12$ had occurred at least once and conduct a simulation study to approximate the expected number of dice rolls that are needed. Use antithetic variates in a simulation to estimate the expected number of dice rolls when using $N = 10^i$ realizations for $i = 2, \cdots, 6$. Compare the variance of your estimator when using antithetic variates with the variance of your previous results obtained without the use of a variance reduction technique.

5.1 Code

```
#include<iostream>
#include<cmath>
#include <tuple>
using namespace std;

tuple<double,double> twoDices(){
    double u1,u2,u1A,u2A ; //u1,u2~u(0,1)
    int n1,n2,n1A,n2A;     //n1,n2~{1,2,3,4,5,6}
    int outcome,outcomeA;   //n1+n2, n1A+n2A
    int outComes[11]  = {2,3,4,5,6,7,8,9,10,11,12}; //outcomes
    int outComesA[11] = {2,3,4,5,6,7,8,9,10,11,12}; //outcomes
    int outComeSum,outComeSumA;//indicator whether all possible outcomes are shown up
    int const maxSum = 77; // sum of all outcomes
    double i,j;

    outComeSum = 0;
    i = 0;
    outComeSumA = 0;
    j = 0;
    while((outComeSum != maxSum) ||(outComeSumA != maxSum)){

        u1 = ((double) rand() / (RAND_MAX));
        u2 = ((double) rand() / (RAND_MAX));
        u1A = 1 - u1;
        u2A = 1 - u2;

        n1 = (int)(u1 * 6.0) + 1 ;
        n2 = (int)(u2 * 6.0) + 1 ;
        n1A = (int)(u1A * 6.0) + 1 ;
        n2A = (int)(u2A * 6.0) + 1 ;

        outcome = n1 + n2;
        outcomeA = n1A + n2A;

        if (outComeSum != maxSum) {
            if( outcome == outComes[outcome-2]){
                outComes[outcome-2] = 0;
                outComeSum = outComeSum + outcome;
            }
            i = i + 1;
        }
        if (outComeSumA != maxSum) {
            if( outcomeA == outComesA[outcomeA-2]){
                outComesA[outcomeA-2] = 0;
                outComeSumA = outComeSumA + outcomeA;
            }
            j = j + 1;
        }
    }
    return make_tuple(i,j);
}
tuple<double,double,double,double> getVars(int n){
    double x, y, xA;   // realizations
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of antithetic

    auto results = twoDices();
    x = get<0>(results);
    y = get<1>(results);
    xA = (x + y)/2; //
    mean = x;
    meanA = xA;
    svs = 0;
    svsA = 0;
    for(int i = 1; i < n; i++)
```

```
    {
        auto results = twoDices();
        x = double(get<0>(results));
        y = double(get<1>(results));
        xA = (x+y)/2;
        svs = svs *(i - 1)/i  + (mean - x)*(mean - x)/(i+1);
        mean = mean + (x-mean)/(i+1);

        svsA = svsA *(i - 1)/i  + (meanA - xA)*(meanA - xA)/(i+1);
        meanA = meanA + (xA-meanA)/(i+1);
    }
    return make_tuple(mean, meanA, svs, svsA);
}

int main(){
    int N[5]={100,1000,10000,100000,1000000};
    for(int i=0;i<5;i++){
        srand(3);       //set up the same random seed for comparison
        auto Var = getVars(N[i]);
        cout<<"For N = "<<N[i]<<endl;
        cout<<"Value of the estimate is "<<get<0>(Var)<<endl;
        cout<<"Value of antithetic estimate is "<<get<1>(Var)<<endl;
        cout<<"Variance of normal estimate is "<<get<2>(Var)<<endl;
        cout<<"Variance of antithetic estimate is "<<get<3>(Var)<<endl;
        cout<<endl;
    }
}
```

5.2 Results

```
For N = 100
Value of the estimate is 62.25
Value of antithetic estimate is 62.25
Variance of normal estimate is 1417.93
Variance of antithetic estimate is 1417.93

For N = 1000
Value of the estimate is 61.264
Value of antithetic estimate is 61.264
Variance of normal estimate is 1515.54
Variance of antithetic estimate is 1515.54

For N = 10000
Value of the estimate is 60.9652
Value of antithetic estimate is 60.9652
Variance of normal estimate is 1284.52
Variance of antithetic estimate is 1284.52

For N = 100000
Value of the estimate is 61.1299
Value of antithetic estimate is 61.1299
Variance of normal estimate is 1273
Variance of antithetic estimate is 1273

For N = 1000000
Value of the estimate is 61.1714
Value of antithetic estimate is 61.1714
Variance of normal estimate is 1285.45
Variance of antithetic estimate is 1285.45
```