Assigment 1

Chao Cheng

January 2019

1 Question 1

Suppose that U is uniformly distributed on (0,1). Write a simulation code to estimate $E(U)$ and $Var(U)$ for $N = 10^i$ realizations for $i = 1, 2, \cdots, 6$. Use one of the built-in pseudorandom number generators that is included with your compiler or software environment. Compare your approximation with the exact answers.

1.1 Code

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

int main()
{
    double expectation,variance; // expectation, variance
    double x2_exp,rd;            //squared expectation, tmp variable rd
    int N[6] = {10,100,1000,10000,100000,1000000};  // iter. numbers

    srand(1);       //set up the random seed
    for(int i=0;i<6;i=i+1){

        expectation = 0;// initialize exp, squared x2, var
        x2_exp = 0;
        variance = 0;

        for(int j=0;j<N[i];j=j+1){
            rd = ((double) rand() / (RAND_MAX));
            x2_exp = x2_exp + rd*rd;
            expectation = expectation + rd;
        }
         //since exp is double. N[i] converts implicitly to double.
        expectation = expectation / N[i];
        x2_exp = x2_exp /N[i];
        // var = E(x2)-(E(x))2
        variance = x2_exp - expectation * expectation;
        cout<< "E(x) estimation for N="<<N[i]<<" is "<< expectation << endl;
        cout<< "Var(x) estimation for N="<<N[i] <<" is "<< variance << endl;
        cout<<"\n";
    }
    return 0;
}
```

1.2 Result

```
E(x) estimation for N=10 is 0.443743
Var(x) estimation for N=10 is 0.0959475

E(x) estimation for N=100 is 0.514696
Var(x) estimation for N=100 is 0.0820968

E(x) estimation for N=1000 is 0.49668
Var(x) estimation for N=1000 is 0.0787169

E(x) estimation for N=10000 is 0.500206
Var(x) estimation for N=10000 is 0.0842092

E(x) estimation for N=100000 is 0.500207
Var(x) estimation for N=100000 is 0.0831451

E(x) estimation for N=1000000 is 0.500005
Var(x) estimation for N=1000000 is 0.0832935
```

2 Question 2

Repeat problem 1 but instead use one of the linear congruential generators given in our textbook in Table 2.1 proposed by L'Ecuyer.

```
#include<iostream>
// Table 2.1 by Fishman,
// pg. 46 Monte Carlo Methods in Financial Engineering
#define a 742938285
#define m 2147483647 // 2^31-1
#define q 2          // q = [m/a]
#define r 661607077  // r =m mod a
using namespace std;

int linear_rand(int seed = 1)
{
    static int x = seed; //integer variable holding the current x_i
    int k;
    k = x/q;
    x = a*(x-k*q)-k*r;
    if (x<0) {
        x=x+m;
    }
    return x ;
}

int main()
{
    double expectation,variance; // expectation, variance
    double x2_exp,rd,seed; //squared expectation, tmp variable rd, and seed
    int N[6] = {10,100,1000,10000,100000,1000000};  //iter. numbers

    seed = 1;      //set up the random seed
    for(int i=0;i<6;i=i+1){

        expectation = 0;// initialize exp, squared x2, var
        x2_exp = 0;
        variance = 0;

        for(int j=0;j<N[i];j=j+1){
            rd = ((double) linear_rand(seed)/(m));
            x2_exp = x2_exp + rd*rd;
            expectation = expectation + rd;
        }

        expectation = expectation / N[i];
        x2_exp = x2_exp /N[i];
        variance = x2_exp - expectation * expectation;  // var = E(x2)-(E(x))2
        cout<< "E(x) estimation for N="<<N[i]<<" is "<< expectation << endl;
        cout<< "Var(x) estimation for N="<<N[i] <<" is "<< variance << endl;
        cout<<"\n";
    }
    return 0;
}
```

2.1 Result

```
E(x) estimation for N=10 is 0.471451
Var(x) estimation for N=10 is 0.0915605

E(x) estimation for N=100 is 0.529552
Var(x) estimation for N=100 is 0.0757128

E(x) estimation for N=1000 is 0.49569
Var(x) estimation for N=1000 is 0.0803349

E(x) estimation for N=10000 is 0.50515
Var(x) estimation for N=10000 is 0.0840136

E(x) estimation for N=100000 is 0.500722
Var(x) estimation for N=100000 is 0.0831013

E(x) estimation for N=1000000 is 0.500978
Var(x) estimation for N=1000000 is 0.0831946
```

3 Question 3

For the linear congruential generator you used in problem 2, please complete a "lattice plot" similar to Figure 2.2 in our textbook.

3.1 Code part a

```
#include <iostream>
#include <fstream>
// Fig 2.2,pg. 49 Monte Carlo Methods in Financial Engineering 2003
#define a 16807
#define m 2147483647 // 2^31-1
#define q 127773     // q = [m/a]
#define r 2836   // r =m mod a
using namespace std;

// linear congruential generator
int linear_rand(int seed = 1)
{
    static int x = seed; //integer variable holding the current x_i
    int k;
    k = x/q;
    x = a*(x-k*q)-k*r;
    if (x<0) {
        x=x+m;
    }
    return x ;
}


int main(){
    double ui; // random variables between 0-1
    int seed;
    ofstream myfile;
    seed = 1;
     //save random numbers into a file for later plotting
    myfile.open("points.csv");
    ui = ((double) linear_rand(seed))/(m);
    myfile<<ui<<"\n";
    for(int i=0;i<1000000;i++){
        ui = ((double) linear_rand(seed))/(m);
        myfile<<ui<<"\n";
    }
    myfile.close();
    return 0;
}
```

3.2 Code part b

```
#include <iostream>
#include <fstream>
// Fig.2.2, pg. 49 Monte Carlo Methods in Financial Engineering 2003
#define a 6
#define m 11 // 11
#define q 1  // q = [m/a]
#define r 5  // r =m mod a
using namespace std;

// linear congruential generator
int linear_rand(int seed=1 )
{
    static int x = seed; //integer variable holding the current x_i
    int k;
    x = a*x % 11;  //since m is not too large.
    if (x<0) {
        x=x+m;
    }
    return x ;
}


int main(){
//    Gnuplot gp;
    double ui; // random variables between 0-1
    int seed;
```

```
    ofstream myfile;
    seed = 1;
    myfile.open("points_b.csv");
    ui = ((double) linear_rand(seed))/(m);
    myfile<<ui<<"\n";
    for(int i=0;i<10;i++){
        ui = ((double) linear_rand(seed))/(m);
        myfile<<ui<<"\n";
    }
    myfile.close();
    return 0;
}
```

3.3 Code part c

```
load points_b.csv
ui=points_b(1:10);
uj=points_b(2:11);
subplot(1,2,1);
plot(ui,uj,'r.');
xlabel('u_i');
ylabel('u_{i+1}');

load points.csv
ui=points(1:1000000);
uj=points(2:1000001);
subplot(1,2,2);
plot(ui,uj,'r.');
xlim([0 0.001])
xlabel('u_i');
ylabel('u_{i+1}');
```

3.4 Result

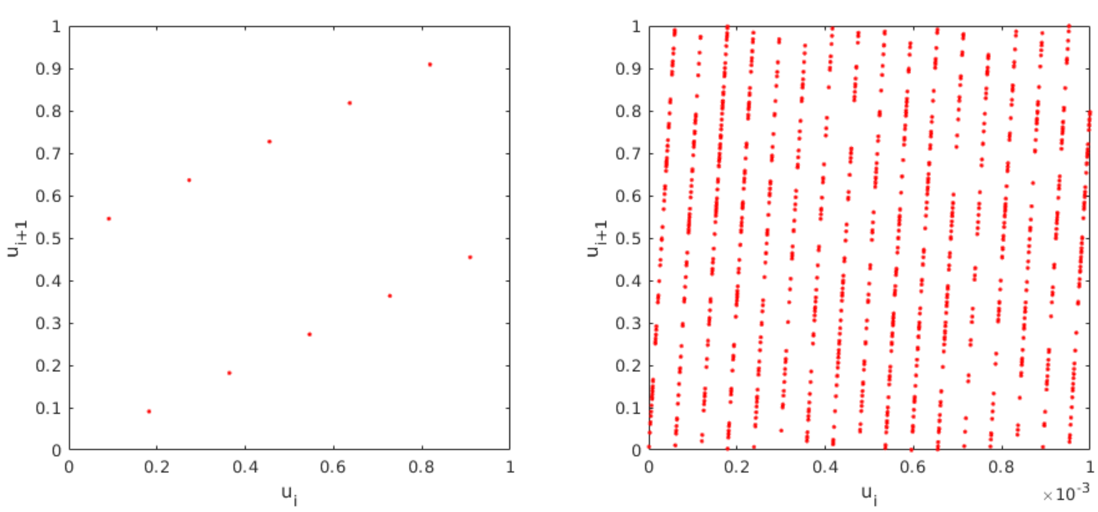

Figure 1: Lattice Plot

4 Question 4

Repeat problem 1 but instead estimate $Cov(U, e^U)$.

4.1 Code

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

int main()
{
    double expectation, covariance; // expectation, covariance
    double eu, u_eu, rd, tmp;  //E(e^u), E(u*e^u), u=rd, tmp to hold e^u
    int N[6] = {10,100,1000,10000,100000,1000000};//array to store all iter. #



    srand(1);       //set up the random seed
    for(int i=0;i<6;i=i+1){

        expectation = 0; // initialize expectation, e^u , u*e^u, and covariance
        eu = 0;
        u_eu = 0;
        covariance = 0;

        for(int j=0;j<N[i];j=j+1){
            rd = ((double) rand() / (RAND_MAX));
            tmp = exp(rd);
            eu = eu + tmp;      //sum e^u
            u_eu = u_eu + rd*tmp; // sum u*e^u
            expectation = expectation + rd;
        }

        expectation = expectation / N[i];
        eu = eu /N[i];
        u_eu = u_eu /N[i];
        covariance = u_eu - expectation * eu;//cov(xy)=E(xy)-E(x)E(y), where y=e^u
        cout<< "E(u)␣estimation␣for␣N="<<N[i]<<"␣is␣"<< expectation << endl;
        cout<< "Cov(u,e^u)␣estimation␣for␣N="<<N[i] <<"␣is␣"<< covariance << endl;
        cout<<"\n";
    }
    return 0;
}
```

4.2 Result

```
E(u) estimation for N=10 is 0.443743
Cov(u,e^u) estimation for N=10 is 0.15221

E(u) estimation for N=100 is 0.514696
Cov(u,e^u) estimation for N=100 is 0.13914

E(u) estimation for N=1000 is 0.49668
Cov(u,e^u) estimation for N=1000 is 0.132297

E(u) estimation for N=10000 is 0.500206
Cov(u,e^u) estimation for N=10000 is 0.142454

E(u) estimation for N=100000 is 0.500207
Cov(u,e^u) estimation for N=100000 is 0.140494

E(u) estimation for N=1000000 is 0.500005
Cov(u,e^u) estimation for N=1000000 is 0.140786
```

5 Question 5

For uniformly distributed on (0,1) random variable $U_1, U_2, \cdots$. Let

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

In other words, M is equal to the number of random numbers that must be summed to exceed 1. Estimate $E(M)$ using $N = 10^i$ realizations for $i = 2, \cdots, 6$.

5.1 Code

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

int main()
{
    double E_M, M, sum_u; // variable : E(M), M
    double u;             // u~u(0,1)
    int N[6] = {100,1000,10000,100000,1000000};  //array to store all iter. #
    int counter ;  //count the # of occurrence when the sum exceeds 1

    srand(1);      //set up the random seed
    for(int i=0;i<5;i=i+1){

        M       = 0;      // initialize E(M), M, sum u, counter
        E_M     = 0;
        sum_u   = 0;
        counter = 0;

        for(int j=0;j<N[i];j=j+1){
            if (sum_u <= 1.0) {
                u = ((double) rand() / (RAND_MAX));
                sum_u = sum_u + u;
                M = M + 1;
            }else{
                E_M = E_M + M ;
                counter = counter + 1;
                //restart a summation since the sum exceeds 1
                sum_u = ((double) rand() / (RAND_MAX));
                M = 1;
            }
        }
        if(counter == 0){
            cout<<"No occurence"<<endl;
        }else{
        //since E_M is double. counter converts implicitly to double.
            E_M = E_M / counter;
        }

        cout<< "E(M) estimation for N="<< N[i] <<" is "<< E_M << endl;
        cout<<"\n";
    }
    return 0;
}
```

5.2 Result

```
E(M) estimation for N=100 is 2.72222

E(M) estimation for N=1000 is 2.75207

E(M) estimation for N=10000 is 2.72378

E(M) estimation for N=100000 is 2.72081

E(M) estimation for N=1000000 is 2.71861
```