Assignment 3

Chao Cheng

February 7, 2019

1 Question 1

A pair of fair dice is continually rolled until all possible outcomes 2, 3,$\cdots$, 12 have occurred at least once. Use a simulation study to approximate the expected number of dice rolls that are needed. Use $N = 10^i$ realizations for $i = 2, \cdots, 6$ in your study.

1.1 Code

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

//since the array is short, so we used a naive search method
bool isInArray(int a, int array[11]){
    if(array[a-2] == a){
        return true;
    }else{
        return false;
    }
}
int main(){
    double u1,u2 ; //u1,u2~u(0,1)
    int n1,n2; //n1,n2~{1,2,3,4,5,6}
    int N[5] = {100,1000,10000,100000,1000000};//array to store all iter. #
    int outcomes[11] = {2,3,4,5,6,7,8,9,10,11,12}; //array to store outcomes
    int outComeSum;//indicator whether all possible outcomes are shown up
    int const maxSum = 77; // sum of all outcomes
    int counter = 0;
    int M = 0;
    double ExpRollNum;

    for(int i=0;i<5;i=i+1){
        counter = 0 ;
        outComeSum = 0;
        M = 0;
        ExpRollNum = 0;
        for(int j=0;j<N[i];j=j+1){
            counter = counter + 1;
            u1 = ((double) rand() / (RAND_MAX));
            u2 = ((double) rand() / (RAND_MAX));
            n1 = (int)(u1 * 6.0) + 1 ;
            n2 = (int)(u2 * 6.0) + 1 ;
            //cout<<n1+n2<<endl;
            //if the sum of two dices have not shown before
            if( isInArray(n1+n2,outcomes)) {
                outcomes[n1+n2-2] = 0; // set the tmp to be 0
                outComeSum = outComeSum + n1 + n2;
                //cout<<"at if "<<outComeSum<<endl;
                if(outComeSum == maxSum){ //all possible outcomes are shown up
                    ExpRollNum = ExpRollNum + counter;
                    M = M + 1;
                    counter = 0;
                    outComeSum = 0;
                    //reset outcome cases
                    for(int k= 0;k<11;k++){
                        outcomes[k] = k + 2;
                    }
                }
            }
        }
        if(M==0){
            cout<<"No␣occurence!"<<endl;
        }else{
            ExpRollNum = ((double) ExpRollNum)/M;
            cout<< "Expected␣dice␣rolls␣for␣N="<<N[i]<<"␣is␣"<< ExpRollNum << endl;
        }
        //reset
        for(int k= 0;k<11;k++){
            outcomes[k] = k + 2;
        }
    }
}
```

1.2 Result

```
Expected dice rolls for N=100 is 39.5
Expected dice rolls for N=1000 is 53.25
Expected dice rolls for N=10000 is 58.1221
Expected dice rolls for N=100000 is 61.982
Expected dice rolls for N=1000000 is 61.5557
```

2 Question 2

Give an algorithm for simulating X having probability density function

$$f(x) = \frac{1}{2}(1+x)e^{-x}, 0 < x < \infty. \tag{1}$$

Implement your algorithm and check its validity by plotting a histogram of simulated values and comparing with the theoretical density.

2.1 Code part a

```
#include<iostream>
#include<cmath>
#include<fstream>
#define C 2.0*exp(-0.5)
using namespace std;
//acceptance - rejection, try g(x)=0.5*exp(-0.5x);
//f(x) = 0.5(1+x)exp(-x), 0<x<inf
//therefore, c=1.2130(acceptable)
int main(){
    double u1,u2; //u~u(0,1)
    double temp; // f(y)/c*g(y)
    double x,y;//y is the candidate~exp(1), x is the acceptance
    int N = 1000000;//iter. numbers

    ofstream myfile;
    myfile.open("problem2.csv");
    for(int j = 0; j < N; j++){
        u1 = ((double)rand()/(RAND_MAX));
        u2 = ((double)rand()/(RAND_MAX));
        y = -log(u1)/0.5;
        temp = (1+y)*exp(-0.5*y)/C;
        if(u2<temp){
            x=y;
            myfile<<x<<"\n";
        }
    }
    myfile.close();
    return 0;
}
```

2.2 Code part b

```
load problem2.csv;
figure();
h = histogram(problem2,'Normalization','pdf');%hist. normalized to pdf
hold on;
X=linspace(0.0,10.0,1000);
f = @(x)(0.5*(1+x).*exp(-x));
plot(X,f(X),'r');
axis([0,11,0,0.55])
legend('Histogram␣of␣simulated␣values','theoretical␣distribution')
```

2.3 Result

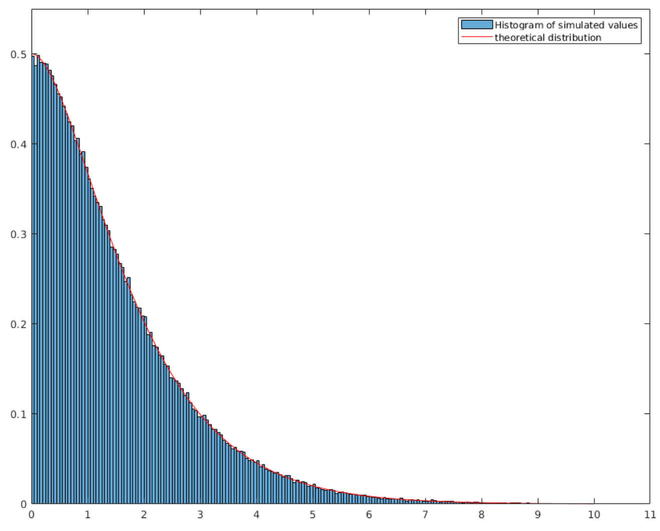

Figure 1: Comparison between normalized histogram and the pdf

3 Question 3

Give an algorithm for simulating X having probability density function

$$f(x) = 30(x^2 - 2x^3 + x^4), 0 \leq x \leq 1. \tag{2}$$

Implement your algorithm and check its validity by plotting a histogram of simulated values and comparing with the theoretical density.

3.1 Code part a

```
#include<iostream>
#include<cmath>
#include<fstream>
#define C 1.875
using namespace std;
//acceptance - rejection, try g(x)=1;
//f(x) = 30*(x^2-2x^3+x^4), 0<x<1
//therefore, c=1.875(acceptable)
int main(){
    double u1,u2; //u~u(0,1)
    double temp; // f(y)/c*g(y)
    double x,y,y2,y3;//y is the candidate, x is the acceptance
    int N = 1000000;//iter. numbers

    ofstream myfile;
    myfile.open("problem3.csv");
    for(int j = 0; j < N; j++){

        u1 = ((double)rand()/(RAND_MAX));
        u2 = ((double)rand()/(RAND_MAX));
        y = u1;
        y2 = y*y;
        y3 = y2*y;
        temp = 30.0*(y2-2*y3+y3*y)/C;
        if(u2<temp){
            x=y;
            myfile<<x<<"\n";
        }
    }
    myfile.close();
    return 0;
}
```

3.2 Code part b

```
load problem3.csv;
figure();
h = histogram(problem3,'Normalization','pdf');%hist. normalized to pdf
hold on;
X=linspace(0.0,1.0,1000);
f = @(x)(30*(x.*x-2*x.^3+x.^4));
plot(X,f(X),'r');
legend('Histogram␣of␣simulated␣values','theoretical␣distribution')
```

3.3 Result

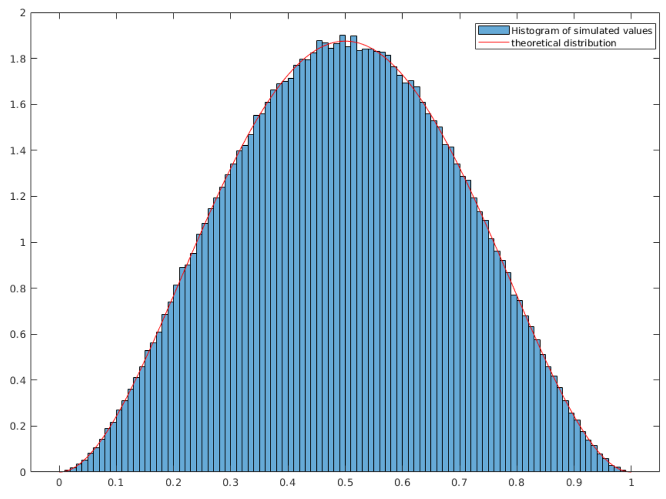

Figure 2: Comparison between normalized histogram and the pdf

4 Question 4

Implement the Box-Muller method and Marsaglia's polar method to simulate samples from a standard Gaussian random variable. Use direct timing results to compare the efficiency of these two approaches. A large number of samples may be required to properly make this comparison.

4.1 Code

```
#include<iostream>
#include<cmath>
#include<chrono>
#define pis 8*atan(1)//2

using namespace std;
using namespace:: chrono;

int N = 10000000;//# of realization
void boxMuller(int n){
    double u1, u2;//u(0,1)
    double x, y; //gsn(0,1)
    for(int i= 0;i<n;i++){
        u1 = ((double)rand()/(RAND_MAX));
        u2 = ((double)rand()/(RAND_MAX));
        x= sqrt(-2*log(u1))*cos(pis*u2);
        y= sqrt(-2*log(u1))*sin(pis*u2);
    }
}
void marsaglia(int n){
    double u1, u2;//u(0,1)
    double v1,v2;//u(-1,1)
    double s,z;
    double x, y; //gsn(0,1)
    for(int i= 0;i<n;i++){
        u1 = ((double)rand()/(RAND_MAX));
        u2 = ((double)rand()/(RAND_MAX));
        v1 = 2*u1 - 1;
        v2 = 2*u2 - 1;
        s = v1*v1 + v2*v2;
        if(s<=1.0){
            z = sqrt(-2*log(s)/s);
            x = z*v1;
            y = z*v2;
        }
    }
}
int main(){
    cout<<"N="<<N<<endl;
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    boxMuller(N);
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    auto duration1 = duration_cast<microseconds>( t2 - t1 ).count();
    cout<<"Box-Muller␣method␣needs␣"<<duration1<<"␣microseconds"<<endl;
    high_resolution_clock::time_point t3 = high_resolution_clock::now();
    boxMuller(N);
    high_resolution_clock::time_point t4 = high_resolution_clock::now();
    auto duration2 = duration_cast<microseconds>( t4 - t3 ).count();
    cout<<"Marsaglias␣method␣needs␣"<<duration2<<"␣microseconds"<<endl;
    if(duration1<duration2){
        cout<<"Box-Muller␣method␣is␣more␣efficient!"<<endl;
    }else{
        cout<<"Marsaglias␣method␣is␣more␣efficient!"<<endl;
    }
    return 0;
}
```

4.2 Result

```
N=10000000
Box-Muller method needs 1706588 microseconds
Marsaglias method needs 1698953 microseconds
Marsaglias method is more efficient!
```

5 Question 5

In class, an algorithm for simulating a Gaussian random variable using rejection from an exponential with rate 1 was given. Mathematically show that, among all exponential distributions, the number of iterations needed for acceptance is minimized when the exponential has rate 1.

5.1 Proof

A standard Gaussian distribution has probability density function

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, 0 < x < \infty. \tag{3}$$

We assume the exponential has a parameter λ,

$$g(x) = \lambda e^{-\lambda x}, 0 < x < \infty. \tag{4}$$

Then, by definition

$$\begin{aligned} c &= \max\left(\tfrac{f(x)}{g(x)}\right) = \max\left(\tfrac{1}{\lambda\sqrt{2\pi}} e^{\lambda x - x^2/2}\right) \\ &= \tfrac{1}{\lambda\sqrt{2\pi}} \max\left(e^{\lambda x - x^2/2}\right) \end{aligned} \tag{5}$$

Therefore, it is equivalent to have

$$\frac{dc}{dx} = \frac{1}{\lambda\sqrt{2\pi}} (\lambda - x) e^{\lambda x - x^2/2} = 0 \tag{6}$$

Hence,

$$x = \lambda. \tag{7}$$

Then,

$$c = \frac{1}{\lambda\sqrt{2\pi}} e^{\lambda^2/2}. \tag{8}$$

In order to have the most optimized algorithm, we need to have c minimized, where the λ is an variable.

$$\frac{dc}{d\lambda} = \frac{1}{\sqrt{2\pi}} (-\lambda^{-2} + 1) e^{\lambda^2/2} = 0 \tag{9}$$

After simplification,

$$\lambda = 1. \tag{10}$$