MA226 : Monte-Carlo Simulation
Linear Congruential Generator

Assignment 3

Turkhade Hrushikesh Pramod
150123044

01-05-2017

1 Problem 1

Our Linear congruential generator has following form:

$x_{i+1} = (a * x_i + b) \bmod m$

$u_{i+1} = \frac{x_{i+1}}{m}$

We are given three different linear congruentials here:

$a = 16807, b = 0, m = 2147483647$

$a = 40692, b = 0, m = 2147483399$

$a = 40014, b = 0, m = 2147483563$

1.1 Source code of the solution

```
ll arr[100010];
int counter[4][100010]={0};

ll big = 2147483647;

ll a_arr[4]={0,16807,40692,40014};
ll m_arr[4]={0,big , 2147483399, 2147483563 };


void genRan(ll seed ,ll length,ll k)
{
        int index=1;
        arr[0]=seed;
        double div=1.0/k;


        for(int j=1;j<=3;++j)
        {
                ll a=a_arr[j];
                ll m=m_arr[j];

                ll q=m/a;
                ll r=m%a;

                for(int i=1;i<=length;i++)
                {
                        ll x = a*(arr[i-1])%m - (arr[i-1]/q)*r;
                        if(x<0)
                        {
                                x=x+m;
                        }
                        arr[i]= x;
                }
                for(int i=0;i<=length;++i)
```

```
                {
                        ll a=floor((arr[i]*1.0/m)/div);
                        counter[j][a+1]++;
                }
        }
}



int main()
{
        cout<<"Enter the length";

        long long int length; cin>>length;

        int seed=7;
        int k=20;

        genRan(seed,length,20);

        //Prints Frequency
        for(int j=1;j<=20;++j)
        {

                cout<<std::setprecision(2)<<fixed;
                cout<<1.0*(j-1)/k<<"-"<<1.0*j/k<<" ";
                for(int i=1;i<=3;i++)
                {
                        cout<<counter[i][j]<<" ";
                }
                cout<<endl;
        }
}
```

1.2 Analysis

1.2.1 For 1000 generated Points

The data frequencies for various intervals for all the three generators for 1000 elements are as follows:

1.2.2 For 10000 generated Points

The data frequencies for various intervals for all the three generators for 1000 elements are as follows:

1.2.3 For 100000 generated Points

The data frequencies for various intervals for all the three generators for 1000 elements are as follows:

1.2.4 Histograms for the above data

- Here, blue denotes the first generator ,green denotes the second and red denotes the third generator.

Tabelle 1: Frequencies for 1000 generated points

Interval	Gen1	Gen2	Gen3
0.00-0.05	53	59	51
0.05-0.10	55	54	58
0.10-0.15	56	45	65
0.15-0.20	58	54	48
0.20-0.25	48	48	50
0.25-0.30	53	51	43
0.30-0.35	50	64	67
0.35-0.40	44	52	43
0.40-0.45	53	47	51
0.45-0.50	49	51	48
0.50-0.55	43	39	36
0.55-0.60	52	48	60
0.60-0.65	47	46	46
0.65-0.70	53	41	56
0.70-0.75	49	47	41
0.75-0.80	48	50	39
0.80-0.85	44	46	52
0.85-0.90	51	62	50
0.90-0.95	49	48	59
0.95-1.00	46	49	38

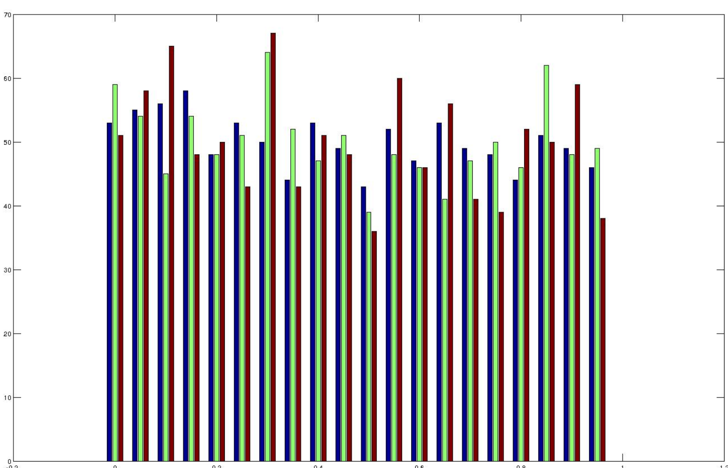

Abbildung 1: Histogram for 1000 generated points

Tabelle 2: Frequencies for 10000 generated points

Interval	Gen1	Gen2	Gen3
0.00-0.05	491	520	515
0.05-0.10	509	487	490
0.10-0.15	523	490	529
0.15-0.20	512	486	493
0.20-0.25	511	485	492
0.25-0.30	479	489	476
0.30-0.35	543	523	557
0.35-0.40	486	522	497
0.40-0.45	526	524	535
0.45-0.50	492	465	472
0.50-0.55	500	504	486
0.55-0.60	492	506	528
0.60-0.65	495	479	492
0.65-0.70	523	489	520
0.70-0.75	492	494	489
0.75-0.80	504	464	497
0.80-0.85	501	505	508
0.85-0.90	494	546	486
0.90-0.95	466	517	461
0.95-1.00	462	506	478

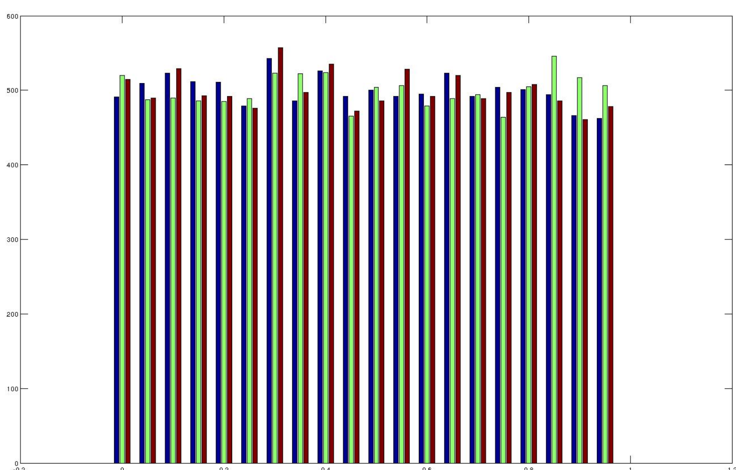

Abbildung 2: Histogram for 10000 generated points

Tabelle 3: Frequencies for 100000 generated points

Interval	Gen1	Gen2	Gen3
0.00-0.05	4984	5065	4878
0.05-0.10	4928	4918	5064
0.10-0.15	5010	5020	5127
0.15-0.20	4997	4947	4987
0.20-0.25	4943	4978	5021
0.25-0.30	4906	4971	4993
0.30-0.35	4979	4955	5066
0.35-0.40	5086	4998	4995
0.40-0.45	4930	5111	4991
0.45-0.50	5053	4960	4991
0.50-0.55	4882	4970	4857
0.55-0.60	5019	5019	5069
0.60-0.65	5012	5025	4955
0.65-0.70	5001	4990	5056
0.70-0.75	5048	4973	5093
0.75-0.80	5221	5066	4896
0.80-0.85	4937	5008	5069
0.85-0.90	4901	4974	5010
0.90-0.95	5008	5035	4932
0.95-1.00	5156	5018	4951

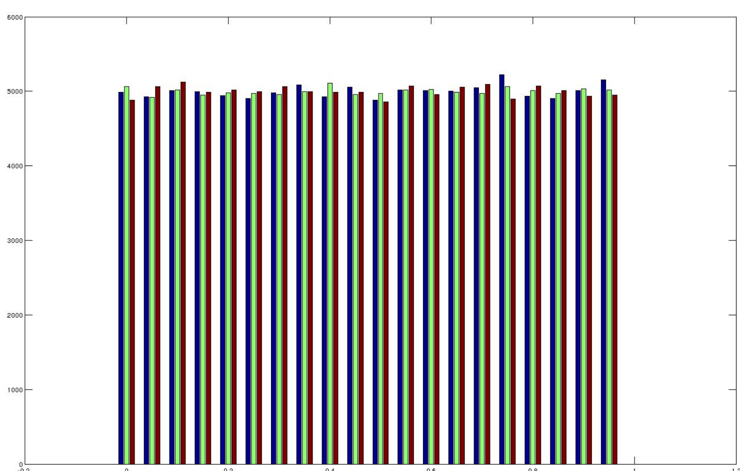

Abbildung 3: Histogram for 100000 generated points

1.3 Plot for $u_i \in [0, 0.0001]$ for $1^s t$ Generator

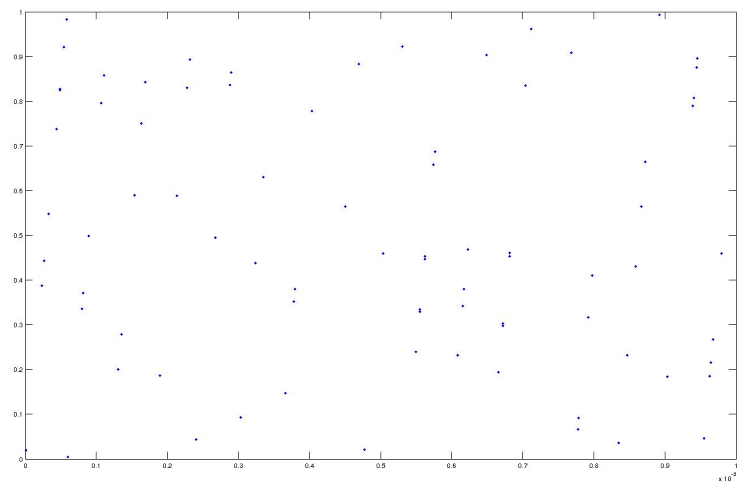

Abbildung 4: Zoomed plot

1.4 Observations

It can be observed that all the three generators are uniform as the amount of numbers generated in each of the interval is close to $\frac{totalpoints}{20}$.

2 Problem 2

Here, we consider an extended fibonacci generator.

$U_i = (U_{i-17} + U_{i-5})$ $modulo$ 2^{31}.

Initially, 17 values are given as a seed using a linear congruential generator.Following generator is used to generate the seeds.

$x_{i+1} = (4062 * x_i)$ $modulo$ 2147483399 with 7 is given as seed to this generator .

2.1 Source code for the solution

```
ll arr[100010];
double normalised[100010];
int counter[100010]={0};

ll mod=2147483648;

void genRan(ll seed ,ll length,ll k)
{
        int index=1;
        arr[0]=seed;
        double div=1.0/k;

        ll a=4062;
        ll m=2147483399;

        ll q=m/a;
        ll r=m%a;

        for(int i=1;i<=length;i++)
        {
                ll x = a*(arr[i-1])%m - (arr[i-1]/q)*r;
                if(x<0)
                {
                        x=x+m;
                }
                arr[i]= x;
        }
        for(int i=0;i<=length;++i)
        {
                ll a=floor((arr[i]*1.0/m)/div);
                counter[a+1]++;
        }
}

void genFib(int length,int k)
{
        genRan(7,17,20);
        double div=1.0/k;

        for(int i=18;i<=length;i++)
        {
                arr[i]=(arr[i-17]+arr[i-5])%mod;
        }
        for(int i=0;i<=length;++i)
        {
                ll a=floor((arr[i]*1.0/mod)/div);
                counter[a+1]++;
        }
}

double calMean(int length)
{
        double total=0;
        for(int i=1;i<=length;++i)
        {
                total+=normalised[i];
        }
```

```
        return total*1.0/length;
}

double calVar(double mean,int length)
{
        double total=0;
        for(int i=0;i<=length;++i)
        {
                total+= (normalised[i]-mean)*(normalised[i]-mean);
        }
        return total/length;
}

void normalise(int length)
{
        for(int i=1;i<=length;++i)
        {
                normalised[i]=arr[i]*1.0/mod;
        }
}

double calACLag(double mean,double var,int l,int length)
{
        double denom = var*length;
        double num=0;
        for(int i=l;i<=length;++i)
        {
                num+=(normalised[i]-mean)*(normalised[i-l]-mean);
        }
        return num/denom;
}

int main()
{
        int length=1000;
        genFib(length,20);
        normalise(length);

        double mean = calMean(length);
        double var = calVar(mean,length);

        cout<<"Mean: "<<mean<<endl;
        cout<<"Variance: "<<var<<endl;

        cout<<"lag 1: "<<calACLag(mean,var,1,length)<<endl;
        cout<<"lag 2: "<<calACLag(mean,var,2,length)<<endl;
        cout<<"lag 3: "<<calACLag(mean,var,3,length)<<endl;
        cout<<"lag 4: "<<calACLag(mean,var,4,length)<<endl;
        cout<<"lag 5: "<<calACLag(mean,var,5,length)<<endl;

        cout<<endl;
}
```

2.2 Plots for (U_i, U_{i+1})

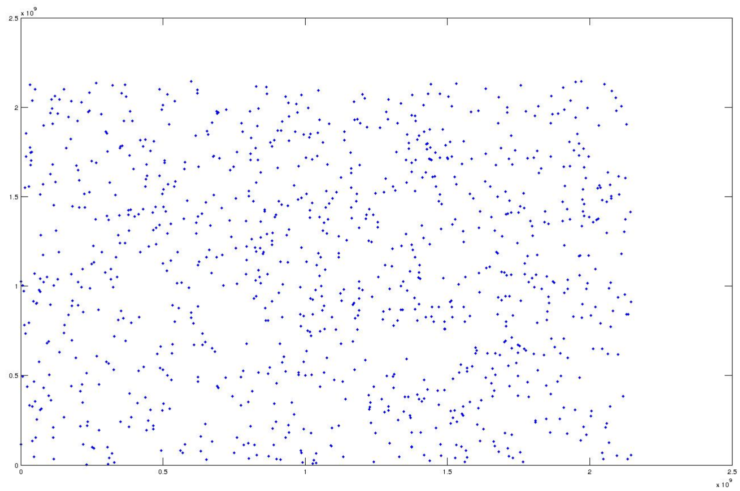

Abbildung 5: Plot of the generator for 1000 points.

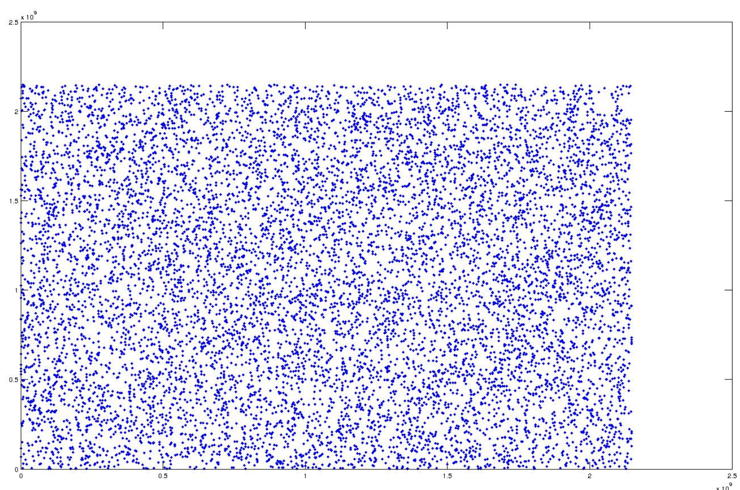

Abbildung 6: Plot of the generator for 1000 points.

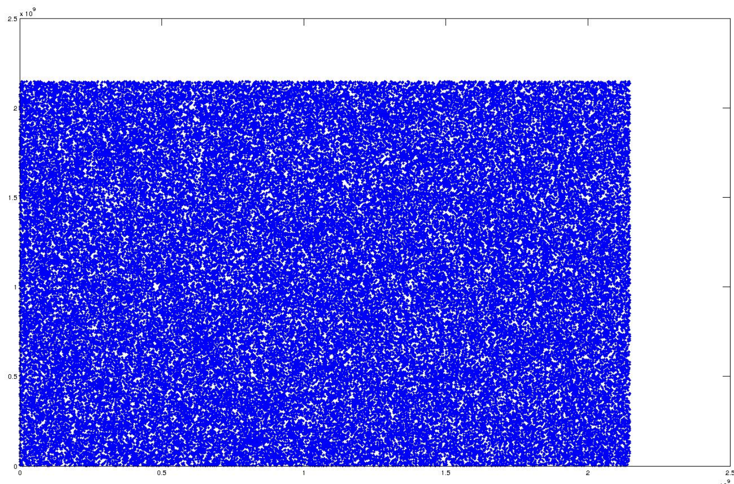

Abbildung 7: Plot of the generator for 1000 points.

2.3 Observations

For a uniform generator following are the expected values of mean and variance.

$E[\mu] = 0.5$

$E[\sigma^2] = 1/12 = 0.083$

Calculated Values:

$\mu = 0.507072$

$\sigma^2 = 0.0806204$

Lags:

$lag - 1 = -0.0156321$

$lag - 2 = 0.025592$

$lag - 3 = 0.0058757$

$lag - 4 = 0.00532893$

$lag - 5 = -0.0312331$