MA 226 - Assignment Report 3

Ayush Sharma
150123046

8 February 2017

Q 1 Implement the linear congruence generator $x_{i+1} = ax_i \mod m$ to generate a sequence x_i and hence uniform random numbers u_i . Make use of the following set of values of a and m: (a) $a = 16807$ and $m = 2^{31} - 1$. (b) $a = 40692$ and $m = 2147483399$. (b) $a = 40014$ and $m = 2147483563$.

Group the values into equidistant ranges for the values of u_i. Tabulate the proportions and draw a bar diagram for the above. What do you observe ? Do it for 1000, 10000 and 100000 values.

For part (a) do the following: Plot the values (u_i, u_{i+1}) on a unit square. Now, zoom into the range $u_i \epsilon [0, 0.001]$. What are your observations ?

Code for C

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void plot(unsigned long long int a, unsigned long long int m, unsigned long long int
     x0, unsigned long long int v0){
   unsigned long long int x = x0, v = 0;
   float u, u1;
   u1 = (x/(float)m);
   do{
      u = u1;
      x = (((a * x) + 0) % m);
      u1 = (x/(float)m);
      printf("(%g,%g)\t", u, u1);
      v++;
   }while(x != x0 && v <= v0);
   printf("\n");
}

void gen(unsigned long long int a, unsigned long long int m, unsigned long long int
    x0, unsigned long long int * f, unsigned long long int v0){
   unsigned long long int x = x0, v = 0; int i = 0;
   do{
      x = (((a * x) + 0) % m);
      i = ((x/(float)m)/0.05);
      if (i == 20) {i--;}
      *(f + i) += 1;
      v++;
```

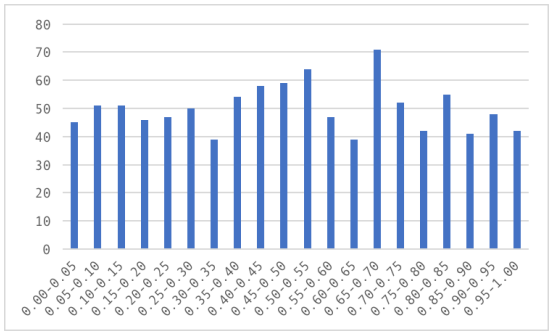
```
27      }while(x != x0 && v <= v0);
28
29  //Printing the frequency table:
30      for(int j = 0; j < 20; j++){
31          printf("\t\t%.2f-%.2f\t:\t%llu\n", (j/(float)20), ((j+1)/(float)20), *(f+j));
32      }
33  }
34
35  void main(){
36      unsigned long long int x0, f[3][3][20] = {0};
37
38  //Frequency Distributuions:
39      printf("Frequency Distributions :");
40  //Part (a)
41      printf("\nFor a = 16807, m = 2^(31) - 1, and x0 = 1 :");
42      printf("\n\t#For 1000 values ::\n");
43      gen(16807, (pow(2,31) - 1), 1, *(*(f+0)+0), 1000);
44
45      printf("\n\t#For 10000 values ::\n");
46      gen(16807, (pow(2,31) - 1), 1, *(*(f+0)+1), 10000);
47
48      printf("\n\t#For 100000 values ::\n");
49      gen(16807, (pow(2,31) - 1), 1, *(*(f+0)+2), 100000);
50
51  //Part (b)
52      printf("\n\nFor a = 40692, m = 214748339, and x0 = 1 :");
53      printf("\n\t#For 1000 values ::\n");
54      gen(40692, 2147483399, 1, *(*(f+1)+0), 1000);
55
56      printf("\n\t#For 10000 values ::\n");
57      gen(40692, 2147483399, 1, *(*(f+1)+1), 10000);
58
59      printf("\n\t#For 100000 values ::\n");
60      gen(40692, 2147483399, 1, *(*(f+1)+2), 100000);
61
62  //Part (c)
63      printf("\n\nFor a = 40014, m = 214748356, and x0 = 1 :");
64      printf("\n\t#For 1000 values ::\n");
65      gen(40014, 2147483563, 1, *(*(f+2)+0), 1000);
66
67      printf("\n\t#For 10000 values ::\n");
```

```
68     gen(40014, 2147483563, 1, *(*(f+2)+1), 10000);
69
70     printf("\n\t#For 100000 values ::\n");
71     gen(40014, 2147483563, 1, *(*(f+2)+2), 100000);
72
73 //Printing (u_i, u_i+1)
74     printf("Plot data :");
75     printf("\nFor a = 16807, m = 2^(31) - 1, and x0 = 1 :\n\tThe values (u_i, u_i+1)
           are :: ");
76     plot(16807, (pow(2,31) - 1), 1, 100000);
77 }
```

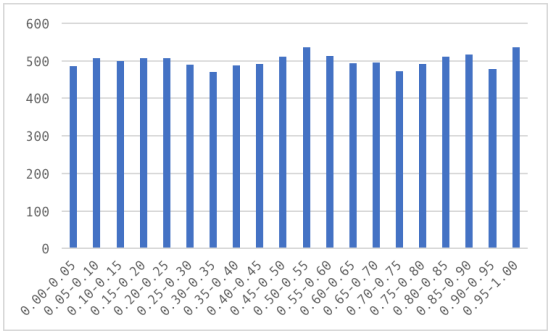
Result:

The histograms can be shown as :

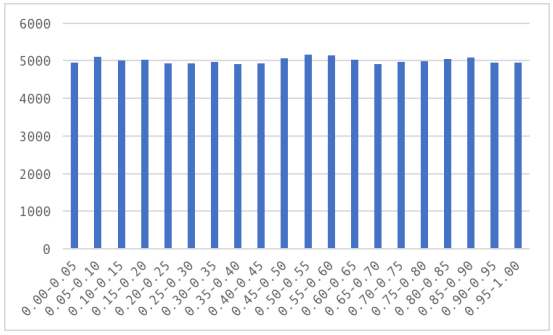
For $a = 16807$, $m = 2^{31} - 1$, and $x_0 = 1$:



(a) ξ_1



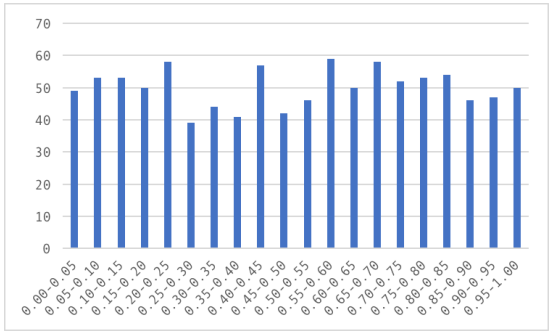
(b) ξ_2



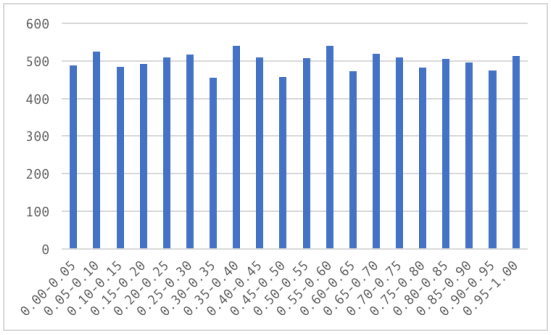
(c) ξ_3

Figure 1: *Histograms for (a) n = 1000, (b) n = 10000 and (c) n = 100000*

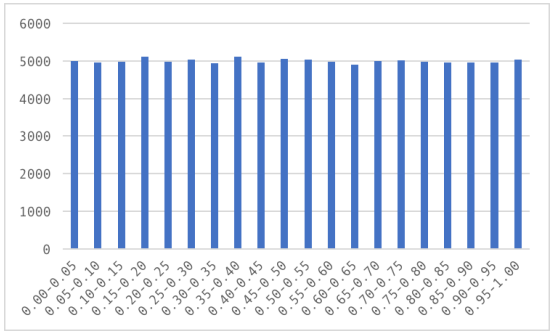
For $a = 40692$, $m = 214748339$, and $x_0 = 1$:



(a) ξ_1



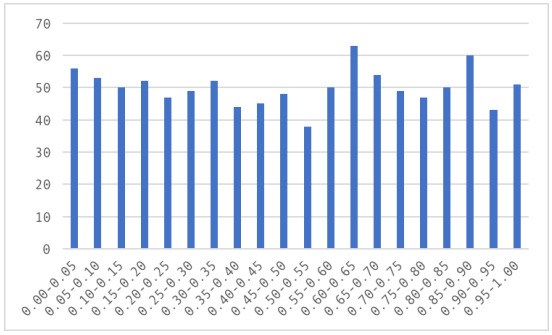
(b) ξ_2



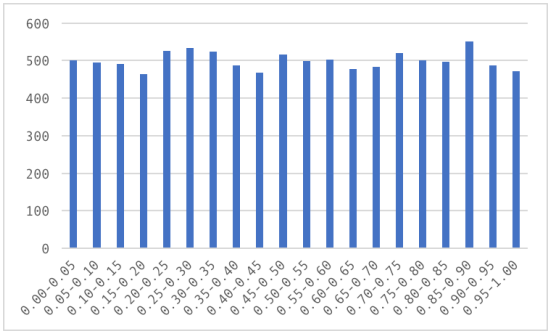
(c) ξ_3

Figure 2: *Histograms for (a) $n = 1000$, (b) $n = 10000$ and (c) $n = 100000$*

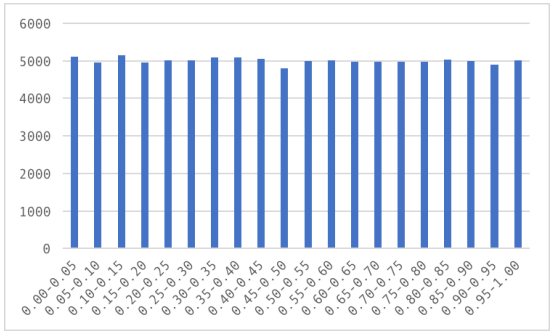
For $a = 40014$, $m = 214748356$, and $x_0 = 1$:



(a) ξ_1



(b) ξ_2

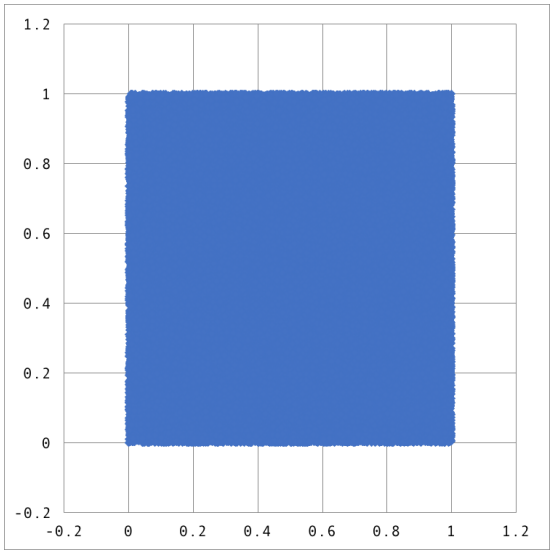


(c) ξ_3

Figure 3: *Histograms for (a) $n = 1000$, (b) $n = 10000$ and (c) $n = 100000$*

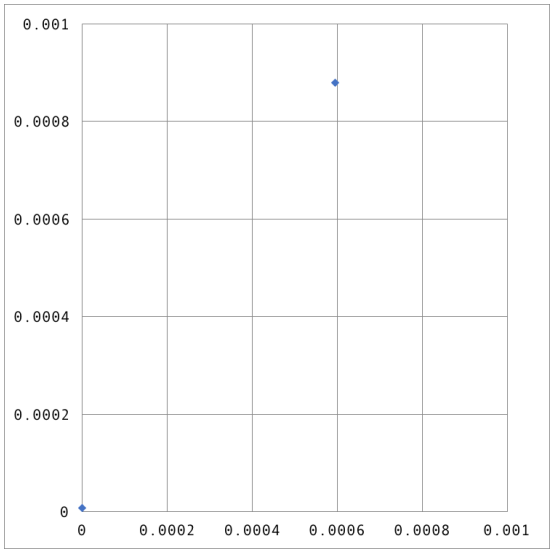
The picture shows that the generated numbers are correct, since on increasing n we are getting the proper shape of uniform distribution.

For $n = 100000$, $a = 16807$, $m = 2^{31} - 1$, and $x_0 = 1$, the plot (u_i, u_{i+1}) is as follows :



(a) ξ_1

Figure 4: *Plot (u_i, u_{i+1})*



(a) ξ_1

Figure 5: *Plot* (u_i, u_{i+1}) *zoomed into the range* $u_i \in [0, 0.001]$

The picture shows that the numbers generated are truly i.i.d. Uniform(0,1), as we see points randomly dispersed in the unit square, in the plot of the sequence (u_{i-1}, ui).
On zooming into the given interval [0, 0.001], we observe that the density of numbers generated is low and distinctively spaced, and the dispersion is not thorough enough.
Although on increasing the number of values generated, the random dispersion will be more efficient, since, the density of the random numbers generated will increase.

Hence, it is concluded that the given random value generator is efficient.

Q 2 Consider the extended Fibonacci generator : $U_i = (U_{i-17} + U_{i-5}) \mod 2^{31}$.

(a) Use the linear congruence generator to generate the first 17 values of U_i. (b) Then generate the values of U_i (say for 1000, 10000 and 100000 values). (c) For each of the above set of values plot (U_i, U_{i+1}). (d) Observe (give the values) the convergence of the sample mean and sample variance towards actual values, and generate a probability distribution with, say, 1000 values generated. (e) Compute the autocorrelation of lags 1, 2, 3, 4, and 5 with 1000 generated values.

Code for C

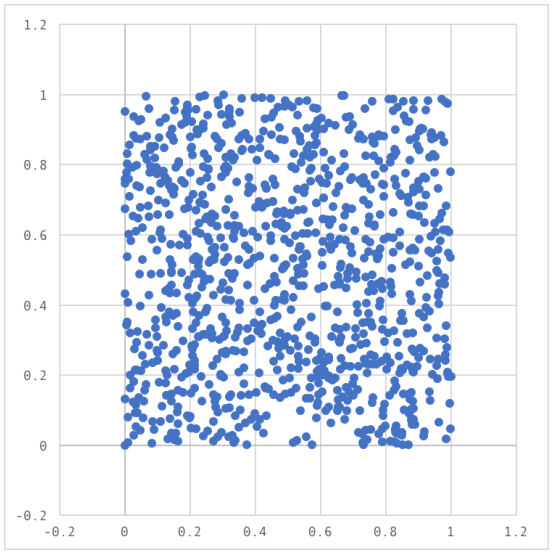
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void list(int v0, float * u){
  long long int x[v0]; int v = 0;
  x[0] = 1;
   u[0] = (x[0]/(float)pow(2,31));
//LCG
   while(v < 17 && v < v0){
    v++;
      x[v] = (((16807 * x[v-1]) + 0) % (long long int)(pow(2,31) - 1));
      u[v] = (x[v]/(float)(pow(2,31) - 1));
  }
//Extended Fibonacci Generator
  while(v < v0){
    v++;
      x[v] = ((x[v-17] + x[v-5]) % (long long int)pow(2,31));
    u[v] = (x[v]/(float)pow(2,31));
   }

}

void data(float * u, int n){
  for (int i = 1; i < n; i++) {
    printf("(%g,%g)\t", u[i-1], u[i]);
  }
}

float mean(float * u, int n){
```

```
31    float m = 0;
32    for (int i = 0; i < n; i++) {
33      m += (u[i]/(float)n);
34    }
35    return m;
36  }
37
38  float variance(float * u, int n, float mu){
39    float v = 0;
40    for (int i = 0; i < n; i++) {
41      v += (pow((u[i]- mu),2)/(float)n);
42    }
43    return v;
44  }
45
46  float rho(float * u, int n, int l, float mu){
47    float num = 0, denom = 0;
48
49    for (int t = l+1; t <= n; t++) {
50      num += ((u[t] - mu) * (u[t - l] - mu));
51    }
52
53    for (int t = 1; t <= n; t++) {
54      denom += pow((u[t]- mu),2);
55    }
56
57    return (num / denom);
58  }
59
60  void main(){
61    float u[1000000];
62    list(100000, u);
63
64    printf("Plot data :");
65    printf("\n\t#For 1000 values ::\n");
66    data(u, 1000);
67
68    printf("\n\t#For 10000 values ::\n");
69    data(u, 10000);
70
71    printf("\n\t#For 100000 values ::\n");
```

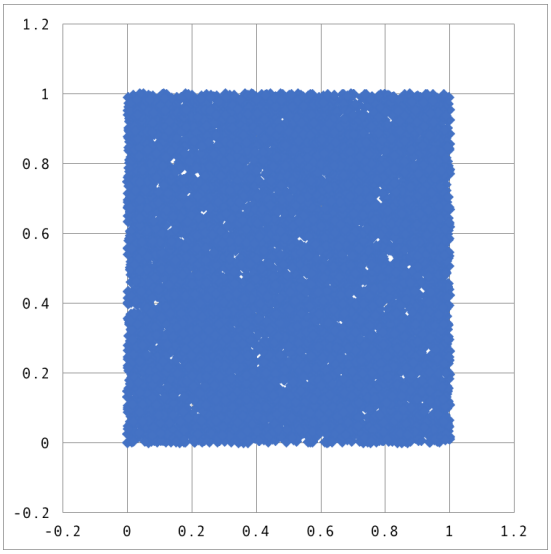
```
72    data(u, 100000);
73    printf("\n");
74
75    float mu = mean(u, 1000);
76    printf("\nThe sample mean for 1000 values is %g.\n", mu);
77
78    float var = variance(u, 1000, mu);
79    printf("\nThe sample variance for 1000 values is %g.\n\n", var);
80
81  //Autocorrelation of lags
82    for (int i = 1; i <= 5; i++) {
83      printf("Autocorrelation of lag %d, with 1000 generated values, is %g.\n", i, rho
            (u, 1000, i, mu));
84    }
85  }
```

Result:

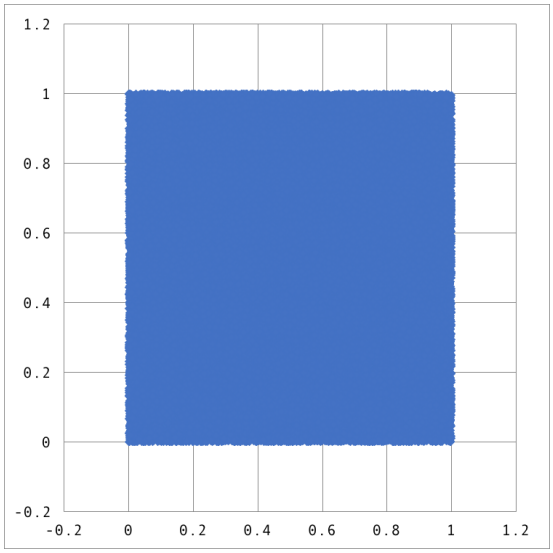
The plot (u_i, u_{i+1}) is as follows :



(a) ξ_1



(b) ξ_2



(c) ξ_3

Figure 6: *Plot* (u_i, u_{i+1}) *for (a)* $n = 1000$*, (b)* $n = 10000$ *and (c)* $n = 100000$

The sample mean and sample variance, calculated for 1000 values, are 0.493665 and 0.0827854 respectively, which are close to theoretical mean $\frac{1}{2}$ i.e. 0.5 and theoretical variance $\frac{1}{12}$ i.e. 0.0833333.

Autocorrelation of lag 1, with 1000 generated values, is -0.0355444.

Autocorrelation of lag 2, with 1000 generated values, is -0.0508289.

Autocorrelation of lag 3, with 1000 generated values, is -0.0167786.

Autocorrelation of lag 4, with 1000 generated values, is 0.0189976.

Autocorrelation of lag 5, with 1000 generated values, is -0.0254554.