

# Monte Carlo Simulation: Assignment 2

---

Yash Vanjani  
Roll No:140123046

26/01/2016

## Question 1

Implement the linear congruence generator  $x_{i+1} = ax_i \bmod m$  to generate a sequence  $x_i$  and hence uniform random numbers  $u_i$ . Make use of the following values of  $a$  and  $m$ : (a)  $a = 16807$  and  $m = 2^{31} - 1$ . (b)  $a = 406932$  and  $m = 214783399$ . (c)  $a = 40014$  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  $u_i \in [0, 0.001]$ . What are your observations?

## Solution

### C++ Code:

```
1 #include <iostream>
2 #include <fstream>
3 #include <cmath>
4
5 using namespace std;
6
7 int main()
8 {
9     ofstream myfile;
10    myfile.open("output.txt", ios::app);
11    long int x[100000];
12    long int a[3];
13    long int m[3];
14    long int q,r,k,b;
15    float u[3][3][100000];
```

```

16  int f[3][3][20];
17  a[0]=16807;
18  a[1]=40692;
19  a[2]=40014;
20  m[0]=(pow(2,31)-1);
21  m[1]=2147483399;
22  m[2]=2147483563;
23  int i,j,l;
24  long int n[3]={1000,10000,100000};
25
26  for(i=0;i<3;i++) //loop for n[i]
27  {
28      for(j=0;j<3;j++) //loop for a[i],m[i]
29      {
30          x[0]=12345;
31          for(l=0;l<n[i]-1;l++) //loop for calculating x[i]
32          {
33              q=m[j]/a[j];
34              r=m[j] % a[j];
35              k=x[l]/q;
36              b=x[l]-(k*q);
37              x[l+1]=(a[j]*b)-(k*r);
38              if (x[l+1]<0)
39                  x[l+1]=x[l+1]+m[j];
40              u[i][j][l+1]=float(x[l+1])/float(m[j]);
41          }
42          u[i][j][0]=float(x[0])/float(m[j]);
43          for(l=0;l<n[i];l++)
44          {
45              if(u[i][j][l]>=0 && u[i][j][l]<0.05)
46                  f[i][j][0]++;
47              if(u[i][j][l]>=0.05 && u[i][j][l]<0.10)
48                  f[i][j][1]++;
49              if(u[i][j][l]>=0.10 && u[i][j][l]<0.15)
50                  f[i][j][2]++;
51              if(u[i][j][l]>=0.15 && u[i][j][l]<0.20)
52                  f[i][j][3]++;
53              if(u[i][j][l]>=0.20 && u[i][j][l]<0.25)
54                  f[i][j][4]++;
55              if(u[i][j][l]>=0.25 && u[i][j][l]<0.30)
56                  f[i][j][5]++;
57              if(u[i][j][l]>=0.30 && u[i][j][l]<0.35)
58                  f[i][j][6]++;
59              if(u[i][j][l]>=0.35 && u[i][j][l]<0.40)
60                  f[i][j][7]++;
61              if(u[i][j][l]>=0.40 && u[i][j][l]<0.45)
62                  f[i][j][8]++;
63              if(u[i][j][l]>=0.45 && u[i][j][l]<0.50)
64                  f[i][j][9]++;
65              if(u[i][j][l]>=0.50 && u[i][j][l]<0.55)
66                  f[i][j][10]++;
67              if(u[i][j][l]>=0.55 && u[i][j][l]<0.60)
68                  f[i][j][11]++;
69              if(u[i][j][l]>=0.60 && u[i][j][l]<0.65)
70                  f[i][j][12]++;
71              if(u[i][j][l]>=0.65 && u[i][j][l]<0.70)
72                  f[i][j][13]++;
73              if(u[i][j][l]>=0.70 && u[i][j][l]<0.75)
74                  f[i][j][14]++;

```

```

75         if(u[i][j][1]>=0.75 && u[i][j][1]<0.80)
76             f[i][j][15]++;
77         if(u[i][j][1]>=0.80 && u[i][j][1]<0.85)
78             f[i][j][16]++;
79         if(u[i][j][1]>=0.85 && u[i][j][1]<0.90)
80             f[i][j][17]++;
81         if(u[i][j][1]>=0.90 && u[i][j][1]<0.95)
82             f[i][j][18]++;
83         if(u[i][j][1]>=0.95 && u[i][j][1]<=1)
84             f[i][j][19]++;
85
86     }
87 }
88
89 }
90
91
92
93 for(i=0;i<3;i++) //loop for n[i]
94 {
95     myfile<<" For n="<<n[i]<<"\n";
96     for(j=0;j<3;j++) //loop for a[i],m[i]
97     {
98         myfile<<"         For a="<<a[j]<<" and m="<<m[j]<<"\n";
99         for(l=0;l<20;l++) //loop for calculating x[i]
100         {
101
102             myfile<<f[i][j][l]<<"\n";
103         }
104     }
105     cout<<"\n";
106
107 }
108 myfile.close();
109
110 myfile.open("2dplot1.txt", ios::app);
111 for(l=0;l<999;l++)
112 {
113     myfile<<u[0][0][l]<<"         "<<u[0][0][l+1]<<"\n";
114 }
115 myfile.close();
116
117 myfile.open("2dplot2.txt", ios::app);
118 for(l=0;l<9999;l++)
119 {
120     myfile<<u[1][0][l]<<"         "<<u[1][0][l+1]<<"\n";
121 }
122 myfile.close();
123
124 myfile.open("2dplot3.txt", ios::app);
125 for(l=0;l<99999;l++)
126 {
127     myfile<<u[2][0][l]<<"         "<<u[2][0][l+1]<<"\n";
128 }
129 myfile.close();
130
131 myfile.open("2dplot_zoom1.txt", ios::app);
132 for(l=0;l<999;l++)
133 {

```

```

134         if(u[0][0][1]<=0.001 )//&& u[0][0][1+1]<=0.001)
135             myfile<<u[0][0][1]<<"          "<<u[0][0][1+1]<<"\n" ;
136     }
137     myfile.close();
138
139     myfile.open("2dplot_zoom2.txt", ios::app);
140     for(l=0;l<9999;l++)
141     {
142         if(u[1][0][1]<=0.001 )//&& u[1][0][1+1]<=0.001)
143             myfile<<u[1][0][1]<<"          "<<u[1][0][1+1]<<"\n" ;
144     }
145     myfile.close();
146
147     myfile.open("2dplot_zoom3.txt", ios::app);
148     for(l=0;l<99999;l++)
149     {
150         if(u[2][0][1]<=0.001 )//&& u[2][0][1+1]<=0.001)
151             myfile<<u[2][0][1]<<"          "<<u[2][0][1+1]<<"\n" ;
152     }
153     myfile.close();
154 }

```

## Output:

```

1 For n=1000
2     For a=16807 and m=2147483647
3 54
4 51
5 41
6 52
7 59
8 45
9 65
10 50
11 58
12 47
13 45
14 49
15 51
16 45
17 54
18 44
19 41
20 49
21 55
22 45
23     For a=40692 and m=2147483399
24 48
25 51
26 54
27 47
28 57
29 68
30 56
31 52
32 52
33 36
34 46
35 32

```

36	56
37	44
38	53
39	49
40	42
41	49
42	54
43	54
44	For a=40014 and m=2147483563
45	51
46	50
47	47
48	49
49	53
50	49
51	50
52	48
53	50
54	41
55	58
56	50
57	35
58	51
59	41
60	54
61	50
62	50
63	59
64	64
65	For n=10000
66	For a=16807 and m=2147483647
67	498
68	502
69	480
70	490
71	474
72	511
73	513
74	535
75	487
76	485
77	470
78	517
79	530
80	501
81	474
82	503
83	478
84	527
85	518
86	507
87	For a=40692 and m=2147483399
88	503
89	507
90	540
91	488
92	486
93	469
94	506

95	505	
96	514	
97	549	
98	504	
99	484	
100	514	
101	477	
102	507	
103	452	
104	485	
105	495	
106	497	
107	518	
108		For a=40014 and m=2147483563
109	470	
110	489	
111	484	
112	530	
113	516	
114	517	
115	526	
116	479	
117	491	
118	518	
119	501	
120	486	
121	471	
122	522	
123	482	
124	499	
125	499	
126	491	
127	521	
128	508	
129		For n=100000
130		For a=16807 and m=2147483647
131	5002	
132	4973	
133	4987	
134	5055	
135	4932	
136	5054	
137	4952	
138	5127	
139	4983	
140	5018	
141	4972	
142	4967	
143	5028	
144	4975	
145	5046	
146	4978	
147	4887	
148	5101	
149	4856	
150	5107	
151		For a=40692 and m=2147483399
152	4990	
153	5112	

154	5140
155	4962
156	4895
157	5070
158	5062
159	4943
160	4997
161	5119
162	4939
163	4979
164	4954
165	4984
166	4911
167	4976
168	5055
169	4967
170	4954
171	4991
172	For a=40014 and m=2147483563
173	4933
174	4889
175	4840
176	5086
177	5037
178	5106
179	5064
180	4889
181	5039
182	4990
183	5126
184	4986
185	4989
186	5010
187	4893
188	4949
189	5114
190	4961
191	5003
192	5096

## Histograms:

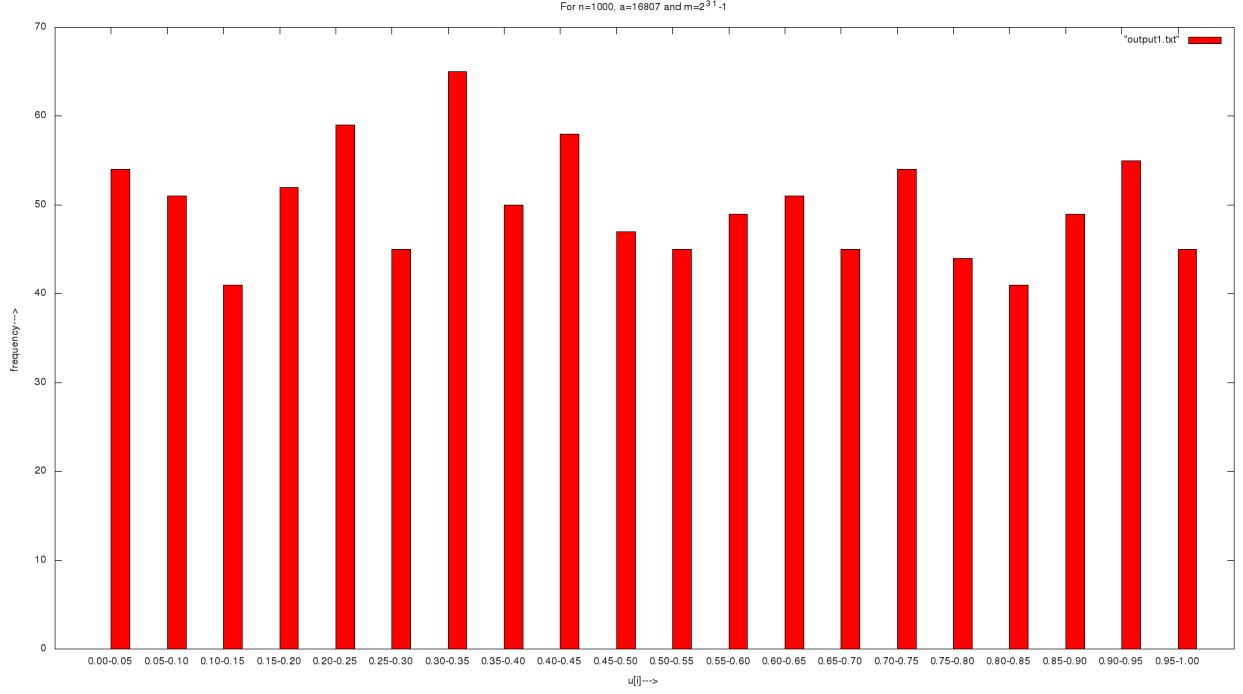


Figure 1: Frequency graph for a=16807,  $m = 2^{31} - 1$  for 1000 random numbers

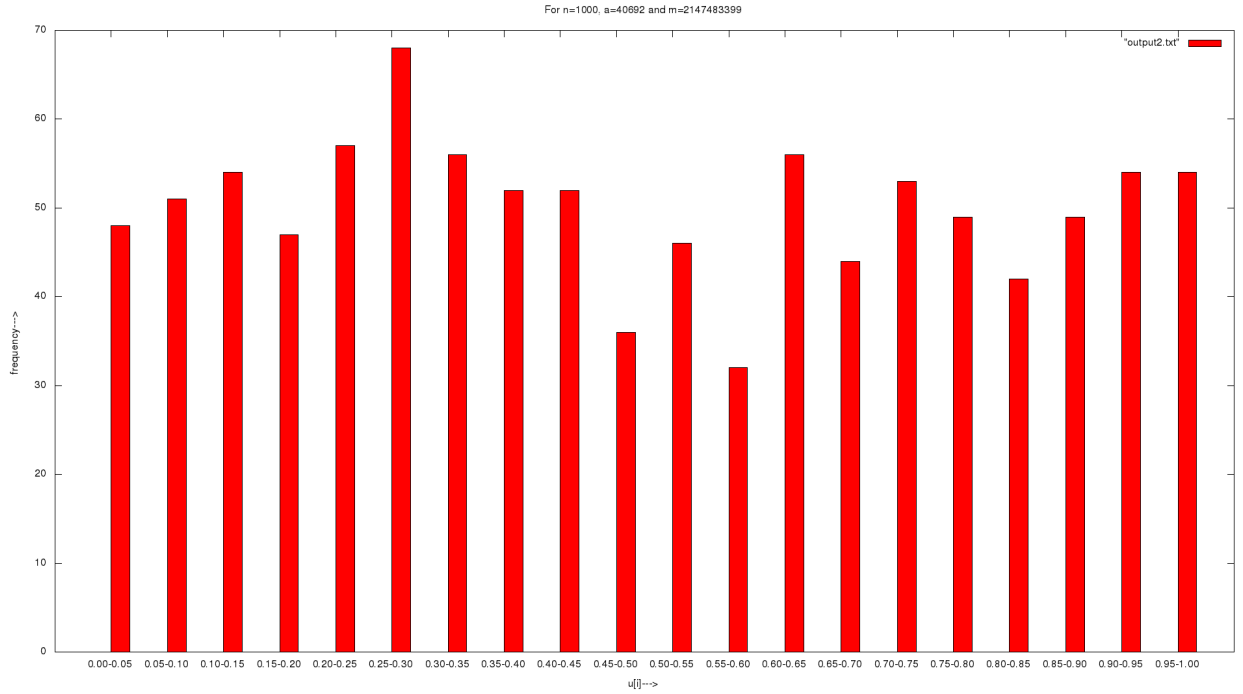


Figure 2: Frequency graph for a=40692,  $m = 2147483399$  for 1000 random numbers





Figure 3: Frequency graph for  $a=40014$ ,  $m = 2147483563$  for 1000 random numbers



Figure 4: Frequency graph for  $a=16807$ ,  $m = 2^{31} - 1$  for 10000 random numbers



Figure 5: Frequency graph for  $a=40692$ ,  $m = 2147483399$  for 10000 random numbers



Figure 6: Frequency graph for  $a=40014$ ,  $m = 2147483563$  for 10000 random numbers

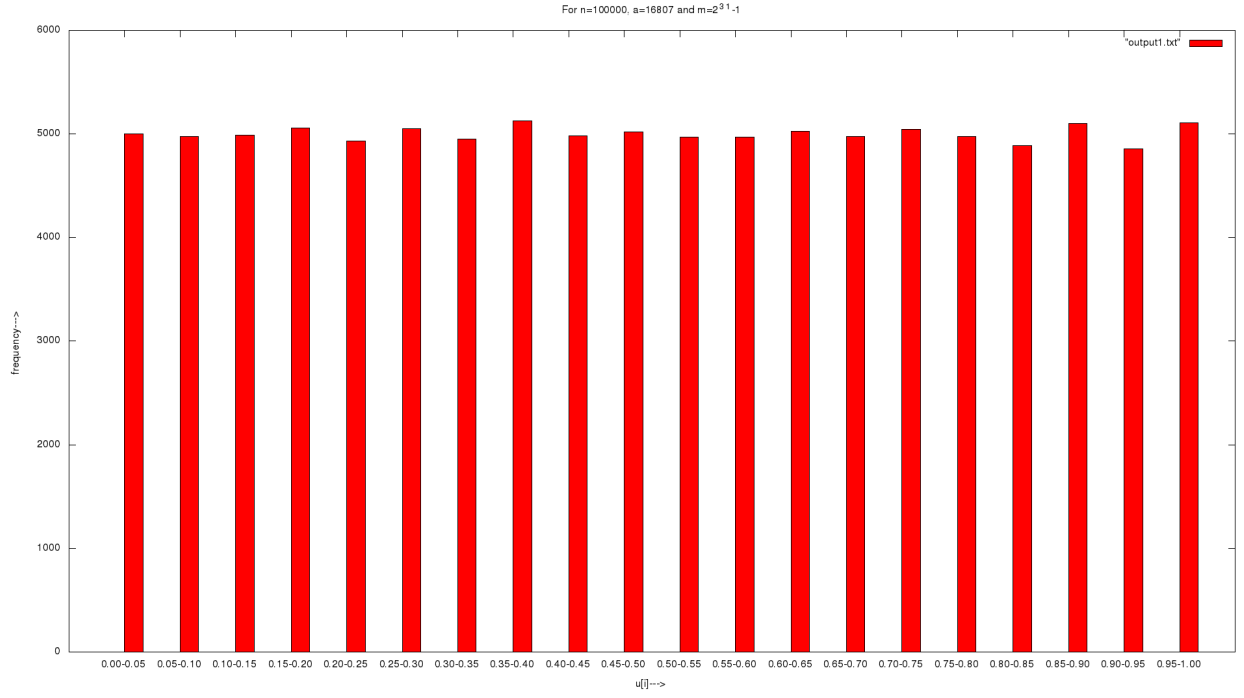


Figure 7: Frequency graph for  $a=16807$ ,  $m = 2^{31} - 1$  for 100000 random numbers



Figure 8: Frequency graph for  $a=40692$ ,  $m = 2147483399$  for 100000 random numbers

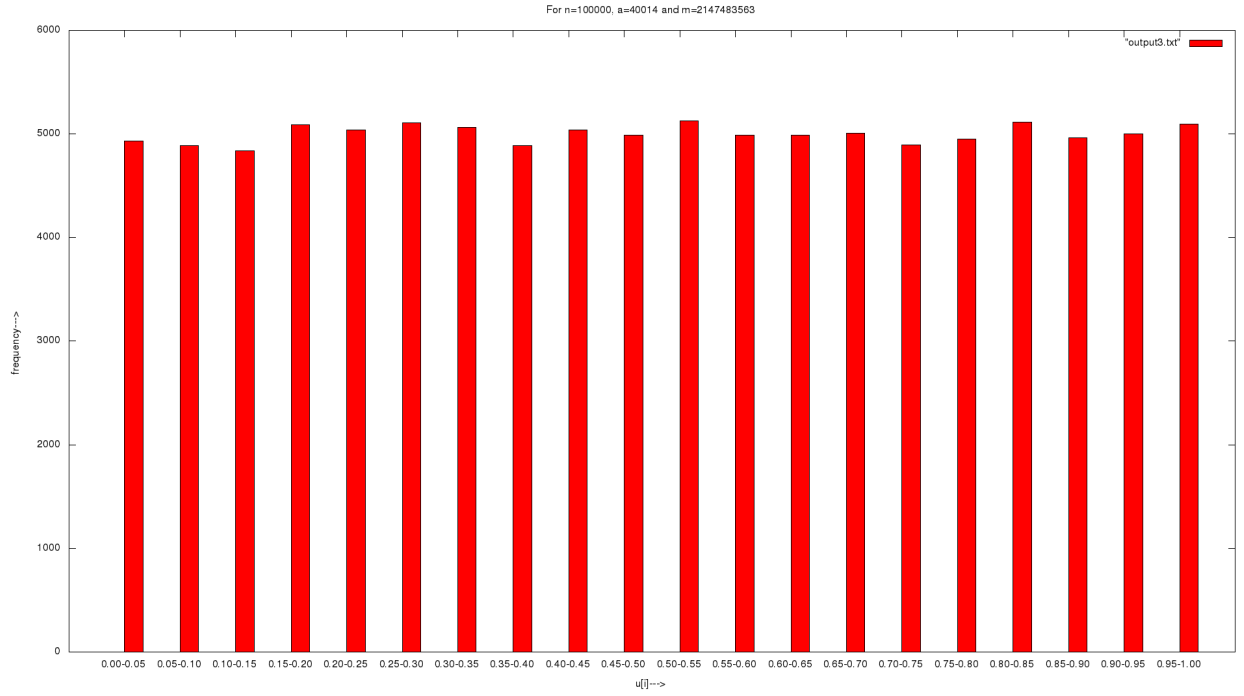


Figure 9: Frequency graph for  $a=40014$ ,  $m = 2147483563$  for 100000 random numbers

### Observations:

- The height of bars in the bar graph tend to equal as the number of random numbers generated are increased thus showing that the random numbers generated become more uniform.

## 2D-Plots:

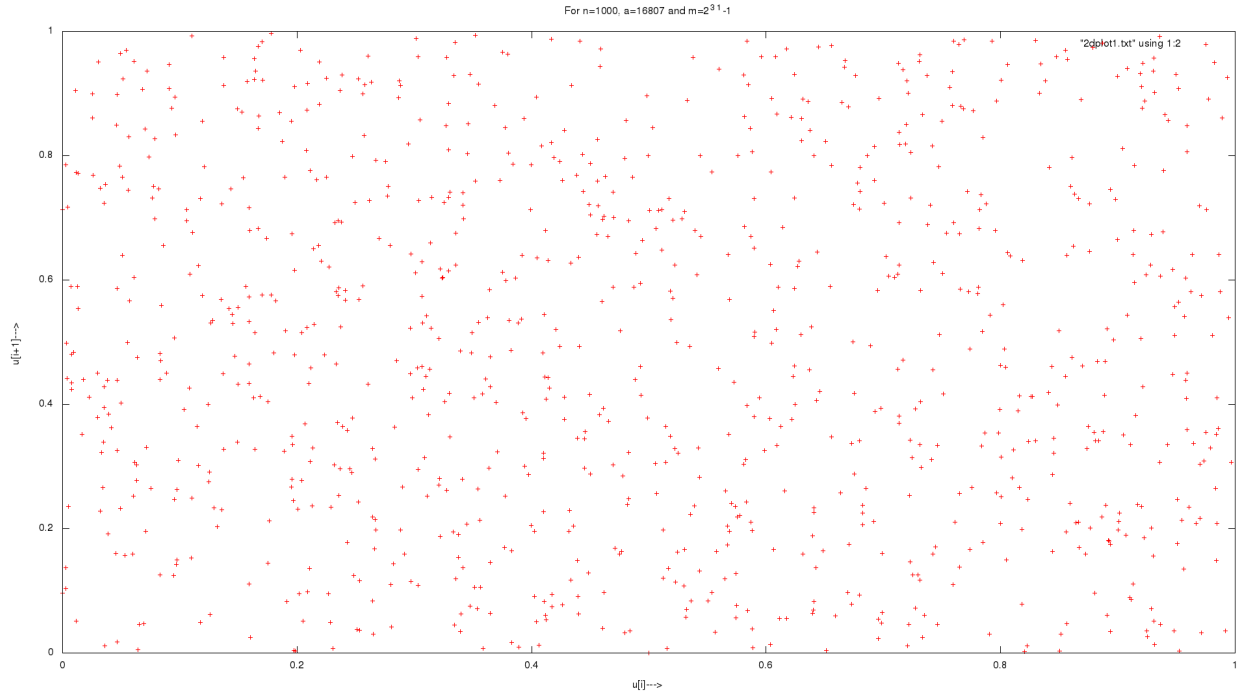


Figure 10: 2D-graph of  $(u_i, u_{i+1})$  for  $a=16807$ ,  $m = 2^{31} - 1$  for 1000 random numbers

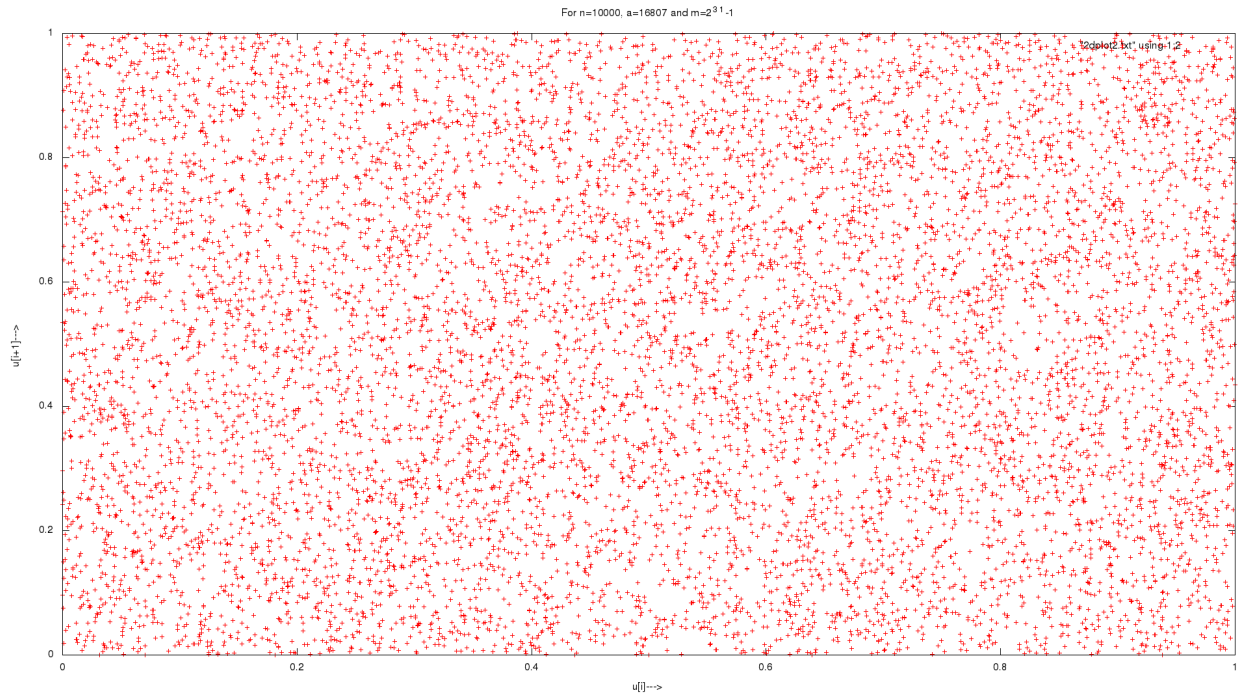


Figure 11: 2D-graph of  $(u_i, u_{i+1})$  for  $a=16807$ ,  $m = 2^{31} - 1$  for 10000 random numbers

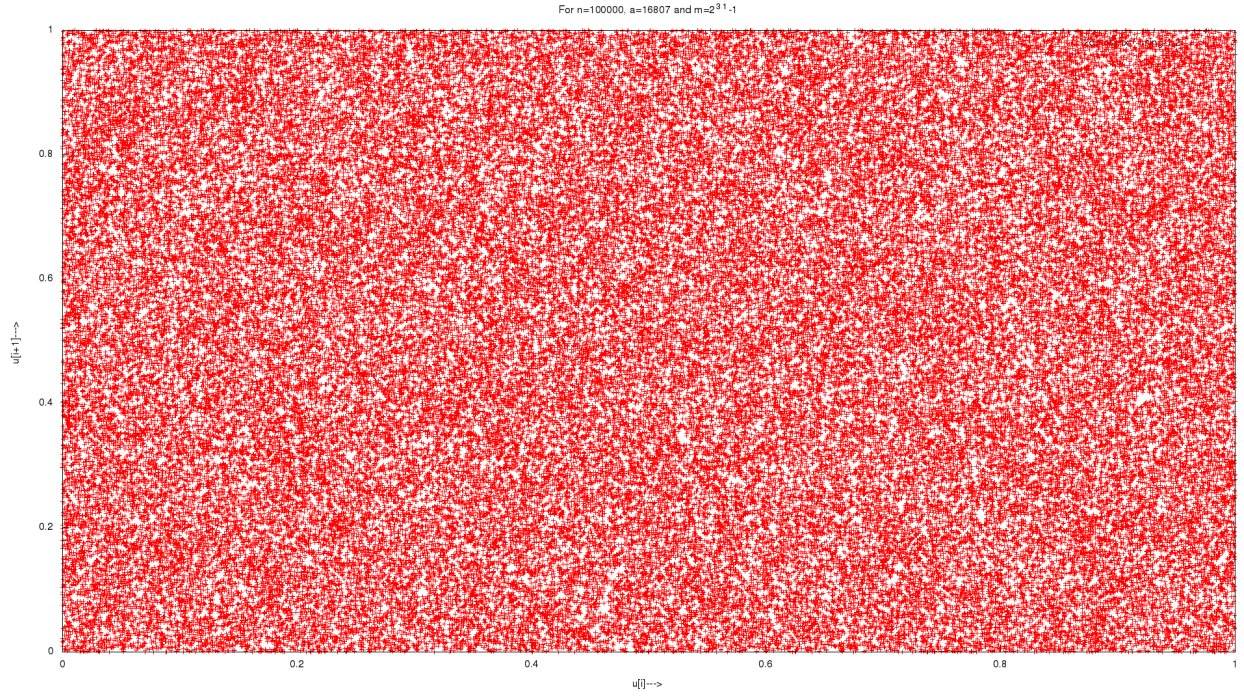


Figure 12: 2D-graph of  $(u_i, u_{i+1})$  for  $a=16807$ ,  $m = 2^{31} - 1$  for 100000 random numbers

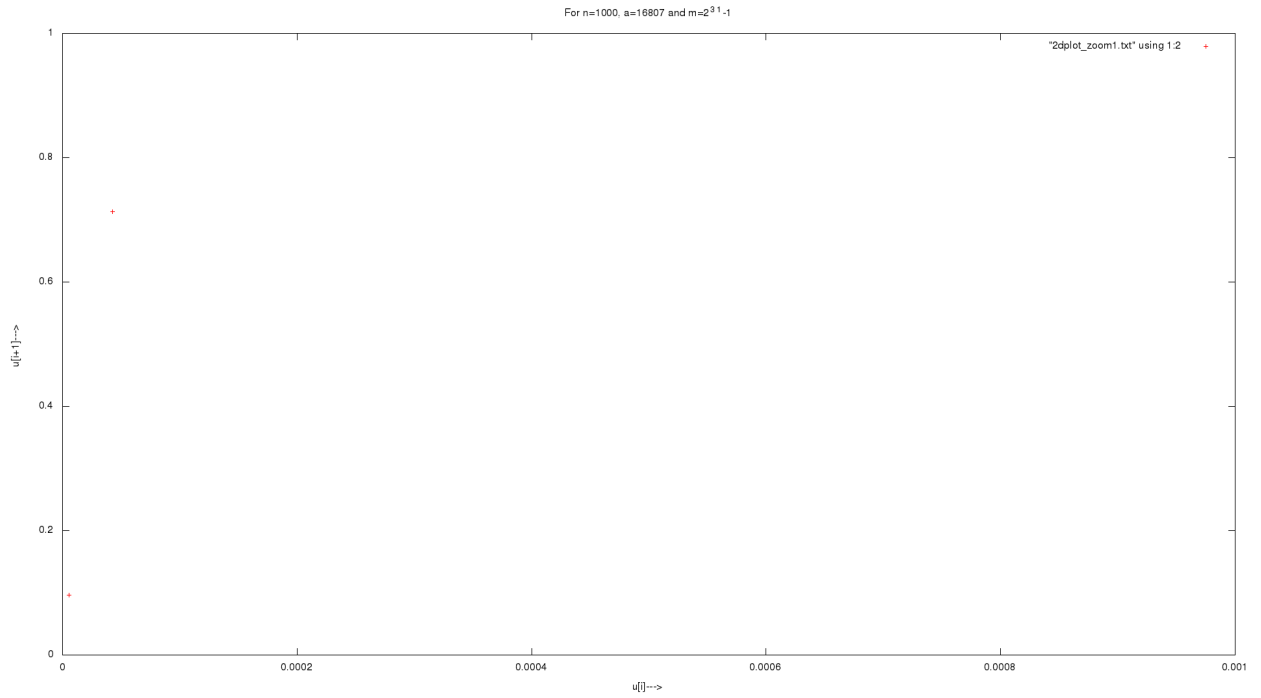


Figure 13: Zoomed  $(u_i, u_{i+1})$  for  $a=16807$ ,  $m = 2^{31} - 1$  for 1000 random numbers

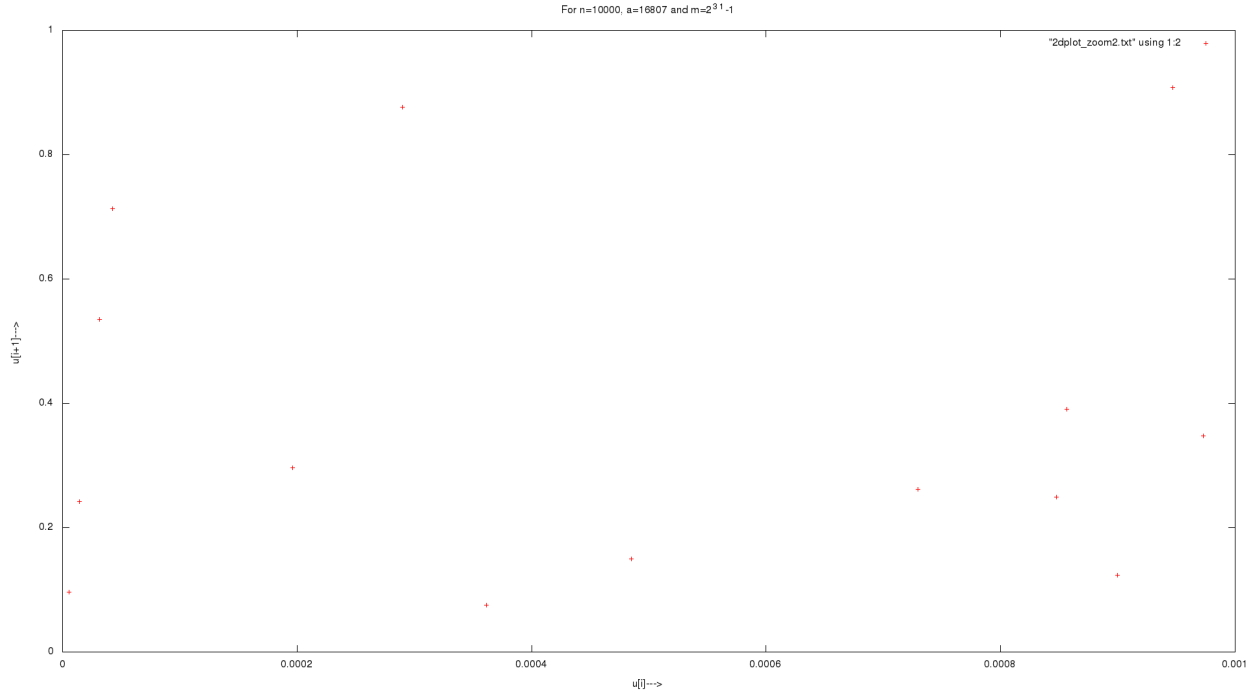


Figure 14: Zoomed  $(u_i, u_{i+1})$  for  $a=16807$ ,  $m = 2^{31} - 1$  for 10000 random numbers

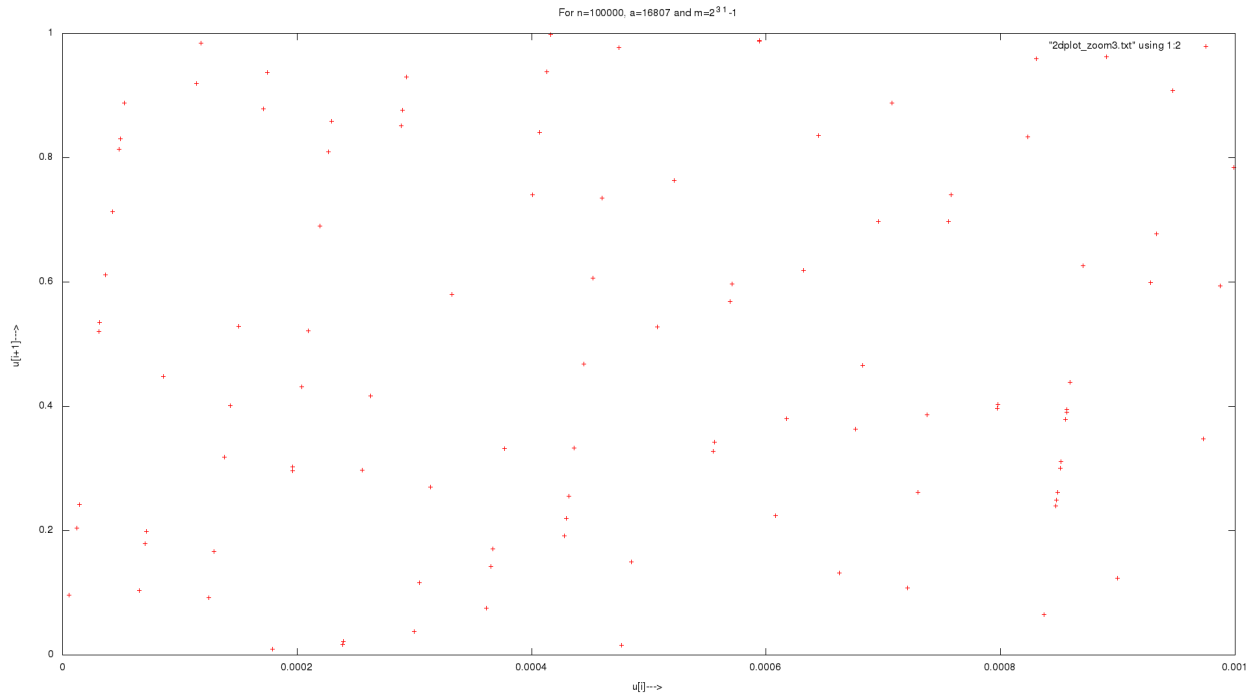


Figure 15: Zoomed  $(u_i, u_{i+1})$  for  $a=16807$ ,  $m = 2^{31} - 1$  for 100000 random numbers

Observations:

- The plot of  $(u_i, u_{i+1})$  got more denser and amount of white gaps in graph decrease as the number of random numbers generated are increased.
- When  $u_i$  is zoomed into the range  $u_i \in [0, 0.001]$ , points are observed to lie on distinct parallel lines as number of random numbers generated are increased.
- These lines are seperated by a constant distance.
- This shows that these pseudo random numbers actually hold a correlation between them.



## Question 2

Consider the extended Fibonacci generator :

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

- (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.

## Solution

### C++ Code:

```
1 #include <iostream>
2 #include <fstream>
3 #include <cmath>
4
5 using namespace std;
6 int main()
7 {
8     ofstream myfile;
9     myfile.open("output.txt");
10    long long int x[3][100000]; //for storing x[i]
11    float u[1000]; //for storing u[i]
12    int f[200];
13    int temp;
14    long long int mean[3][100000];
15    long long int var[3][100000];
16    int l[6]={0,1,2,3,4,5}; //amt of lag
17    float autocorelation[6];
18
19    long long int a,m; //for computing first 17 values of x[i], data taken from
        question 1
20    int i,j;
21    long long int q,r,k,b;
22    x[0][0]=12345;
23    x[1][0]=12345;
24    x[2][0]=12345;
25
26    a=16807;
27    m=2147483399;
28    b=pow(2,31);
29    long long int n[3]={1000,10000,100000};
30
31    //computation part
32
33    for(i=0;i<3;i++) //loop for n[i]
34    {
```

```

35     for (j=0;j<16;j++) //for computing x[i][1] to x[i][16]
36     {
37         q=m/a;
38         r=m%a;
39         k=x[i][j]/q;
40         x[i][j+1]=(a*(x[i][j]-(k*q)))-(k*r);
41         if (x[i][j+1]<0)
42             x[i][j+1]=x[i][j+1]+m;
43     }
44     for (j=16;j<n[i]-1;j++)
45     {
46         x[i][j+1]=((x[i][j-16])+(x[i][j-4]))% b;
47     }
48
49
50     //computing mean
51     mean[i][0]=x[i][0];
52     for (j=1;j<n[i];j++)
53     {
54         mean[i][j]=((mean[i][j-1]*j)+x[i][j])/(j+1);
55     }
56     //computing variance
57     var[i][0]=0;
58     for (j=0;j<n[i]-1;j++)
59     {
60         var[i][j+1]=(((j+1)*(var[i][j]+pow(mean[i][j],2))+pow(x[i][j+1],2)))/(j+2)
61             -pow(mean[i][j+1],2);
62     }
63
64     for (j=0;j<n[i];j++)
65         myfile<<" "<<j<<" "<<x[i][j]<<" "<<mean[i][j]<<" "<<var[i][j]<<"\n";
66     myfile.close();
67
68     //computing frequency and probability distribution
69
70     for (j=0;j<1000;j++)
71     {
72         u[j] = float(x[0][j])/float(b);
73         temp=u[j]/0.005;
74         f[temp]++;
75     }
76
77     for (j=1;j<200;j++)
78         f[j]=f[j]+f[j-1];
79     myfile.open("probability.txt");
80     for (j=0;j<200;j++)
81         myfile<<f[j]<<"\n";
82     myfile.close();
83
84     //computing autocorelation function with lags 1,2,3,4,5
85
86     for (i=0;i<6;i++)
87     {
88         for (j=1[i];j<1000;j++)
89         {
90             autocorelation[i]+=((float(x[0][j])-float(mean[0][999]))*(float(x[0][j-1[i]]-float(mean[0][999]))));
91         }

```

```

92     }
93     myfile.open("autocorrelation.txt");
94     for (i=1; i<6; i++)
95     {
96         autocorrelation[i]=autocorrelation[i]/autocorrelation[0];
97         cout<<"Autocorrelation with lag "<<i<<" for 1000 random numbers = "<<
          autocorrelation[i]<<"\n";
98         myfile<<"Autocorrelation with lag "<<i<<" for 1000 random numbers = "<<
          autocorrelation[i]<<"\n";
99     }
100    myfile.close();
101
102    //printing mean to files
103
104    myfile.open("mean1.txt");
105    for (j=0; j<1000; j++)
106    {
107        myfile<<mean[0][j]<<"\n";
108    }
109    myfile.close();
110
111    myfile.open("mean2.txt");
112    for (j=0; j<10000; j++)
113    {
114        myfile<<mean[1][j]<<"\n";
115    }
116    myfile.close();
117
118    myfile.open("mean3.txt");
119    for (j=0; j<100000; j++)
120    {
121        myfile<<mean[2][j]<<"\n";
122    }
123    myfile.close();
124
125    //printing variance to files
126
127    myfile.open("var1.txt");
128    for (j=0; j<1000; j++)
129    {
130        myfile<<var[0][j]<<"\n";
131    }
132    myfile.close();
133
134    myfile.open("var2.txt");
135    for (j=0; j<10000; j++)
136    {
137        myfile<<var[1][j]<<"\n";
138    }
139    myfile.close();
140
141    myfile.open("var3.txt");
142    for (j=0; j<100000; j++)
143    {
144        myfile<<var[2][j]<<"\n";
145    }
146    myfile.close();
147
148

```

```

149 //printing plot values to files
150
151 myfile.open("2d_plot1.txt");
152 for(i=0;i<999;i++)
153 {
154     myfile<<x[0][i]<<"    "<<x[0][i+1]<<"\n";
155 }
156 myfile.close();
157
158 myfile.open("2d_plot2.txt");
159 for(i=0;i<9999;i++)
160 {
161     myfile<<x[1][i]<<"    "<<x[1][i+1]<<"\n";
162 }
163 myfile.close();
164
165 myfile.open("2d_plot3.txt");
166 for(i=0;i<99999;i++)
167 {
168     myfile<<x[2][i]<<"    "<<x[2][i+1]<<"\n";
169 }
170 myfile.close();
171 }

```

### Output:

```

1 Autocorrelation with lag 1 for 1000 random numbers = 0.00861274
2 Autocorrelation with lag 2 for 1000 random numbers = -0.040237
3 Autocorrelation with lag 3 for 1000 random numbers = 0.0289122
4 Autocorrelation with lag 4 for 1000 random numbers = -0.00226832
5 Autocorrelation with lag 5 for 1000 random numbers = -0.0380737

```

### Observations:

- A low value of autocorrelation shows that the distribution of random numbers is quite uniform.
- Here, an autocorrelation value of 0.0289 shows that the random numbers are quite uniformly distributed, but this distribution could have been more uniform for some other values of the parameters of the random number generator.

## Graphs:

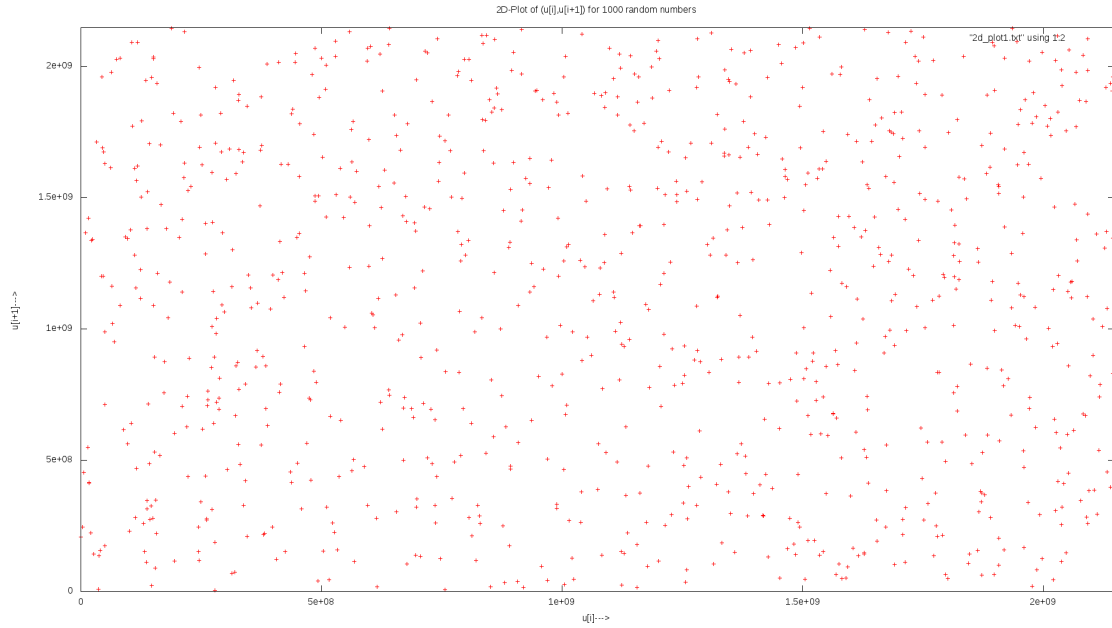


Figure 16: 2D-graph of  $(u_i, u_{i+1})$  for 1000 random numbers

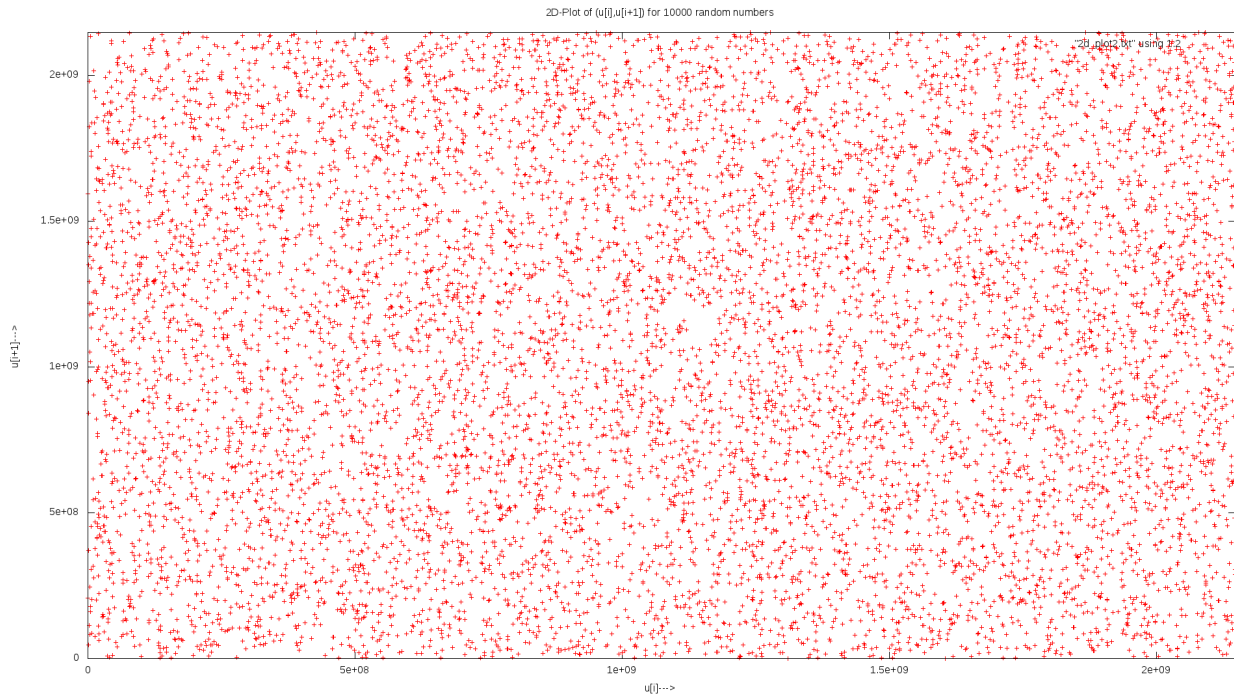


Figure 17: 2D-graph of  $(u_i, u_{i+1})$  for 10000 random numbers

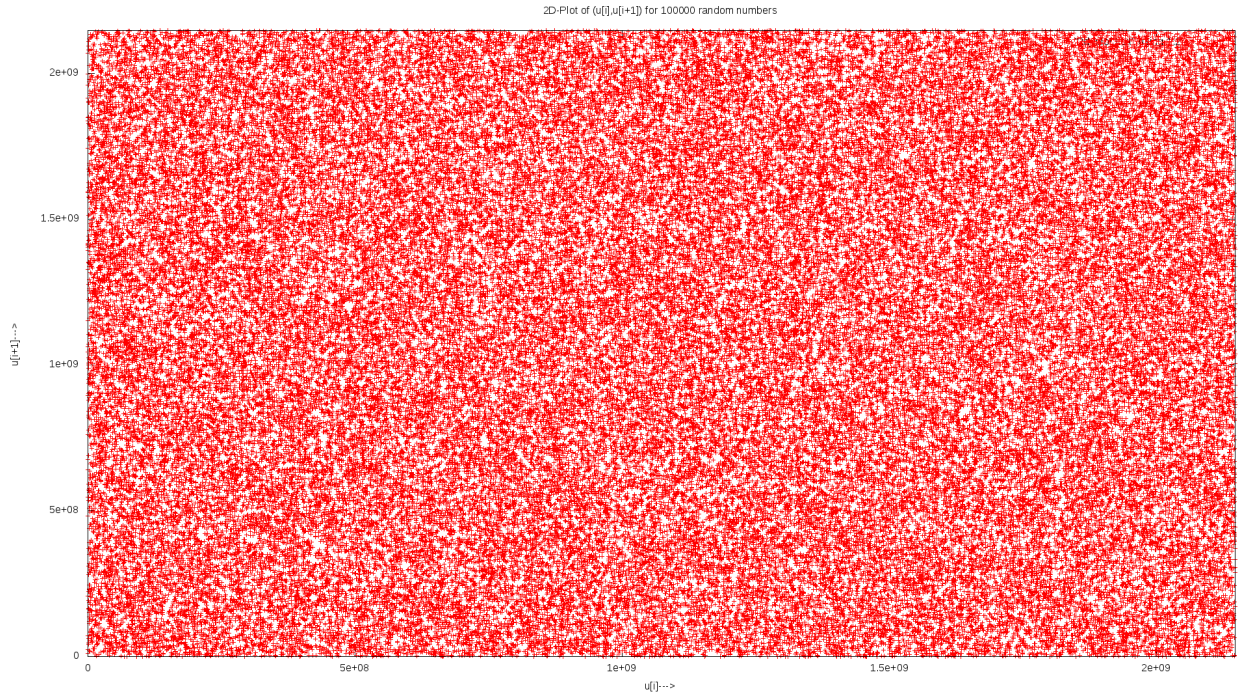


Figure 18: 2D-graph of  $(u_i, u_{i+1})$  for 100000 random numbers

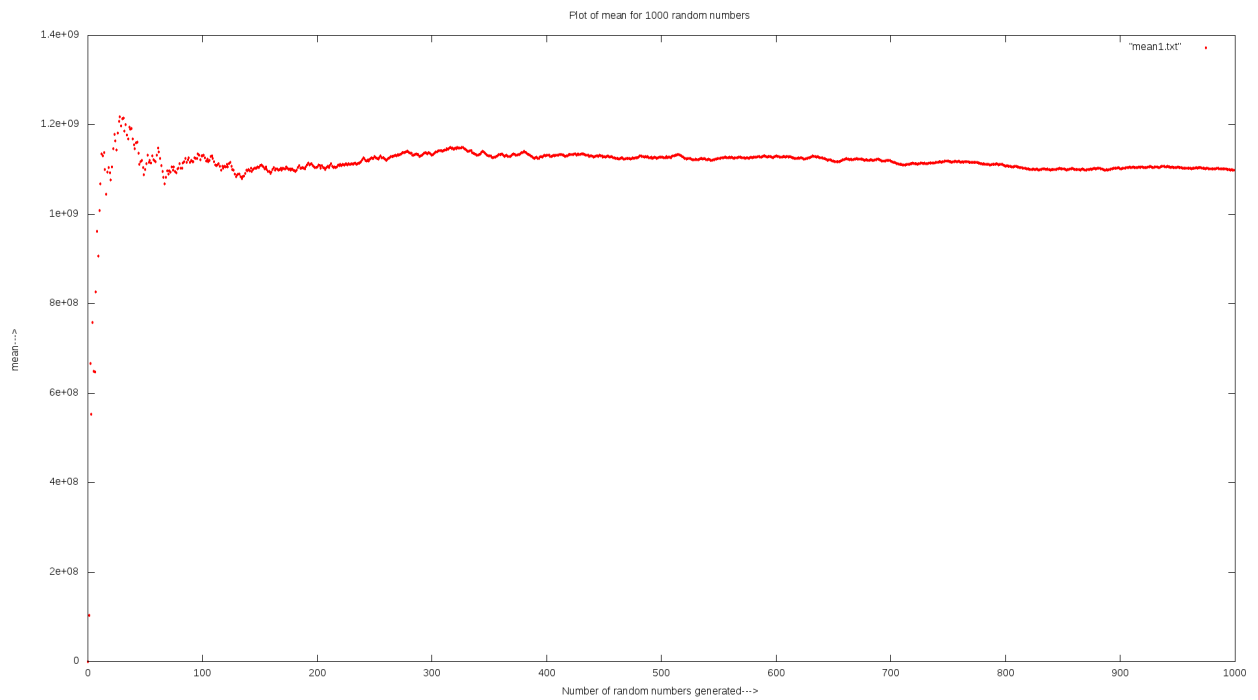


Figure 19: Mean for 1000 random numbers

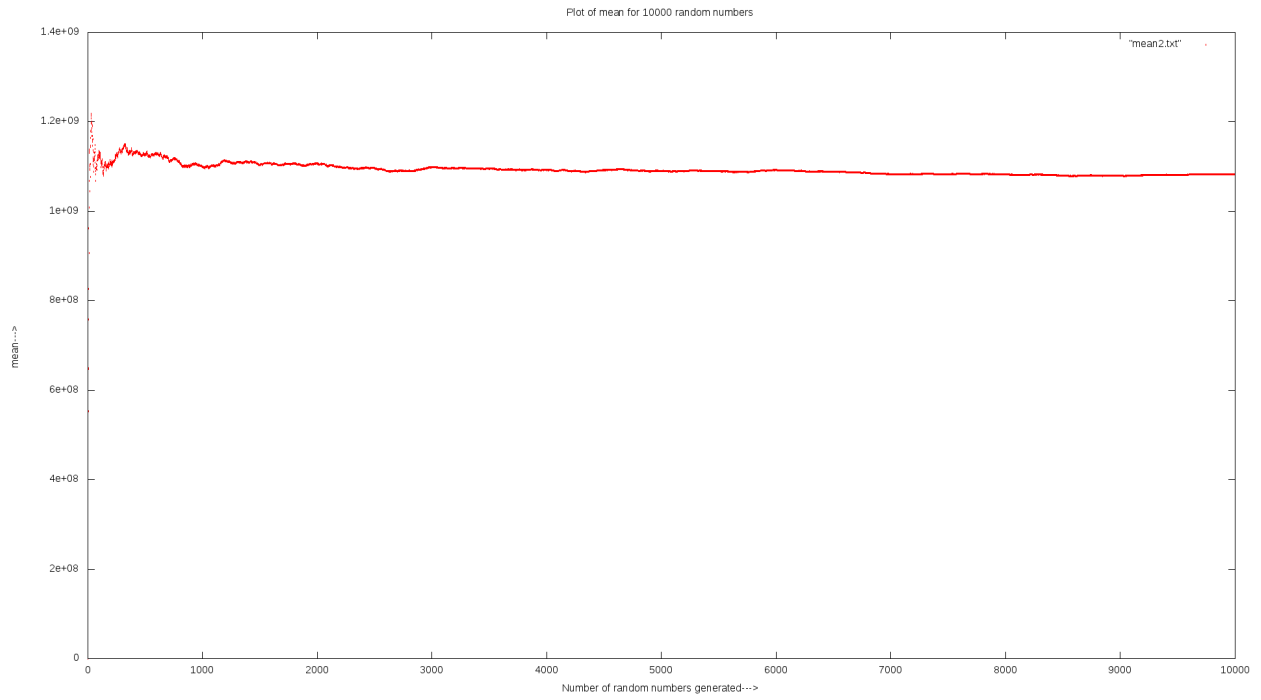


Figure 20: Mean for 10000 random numbers

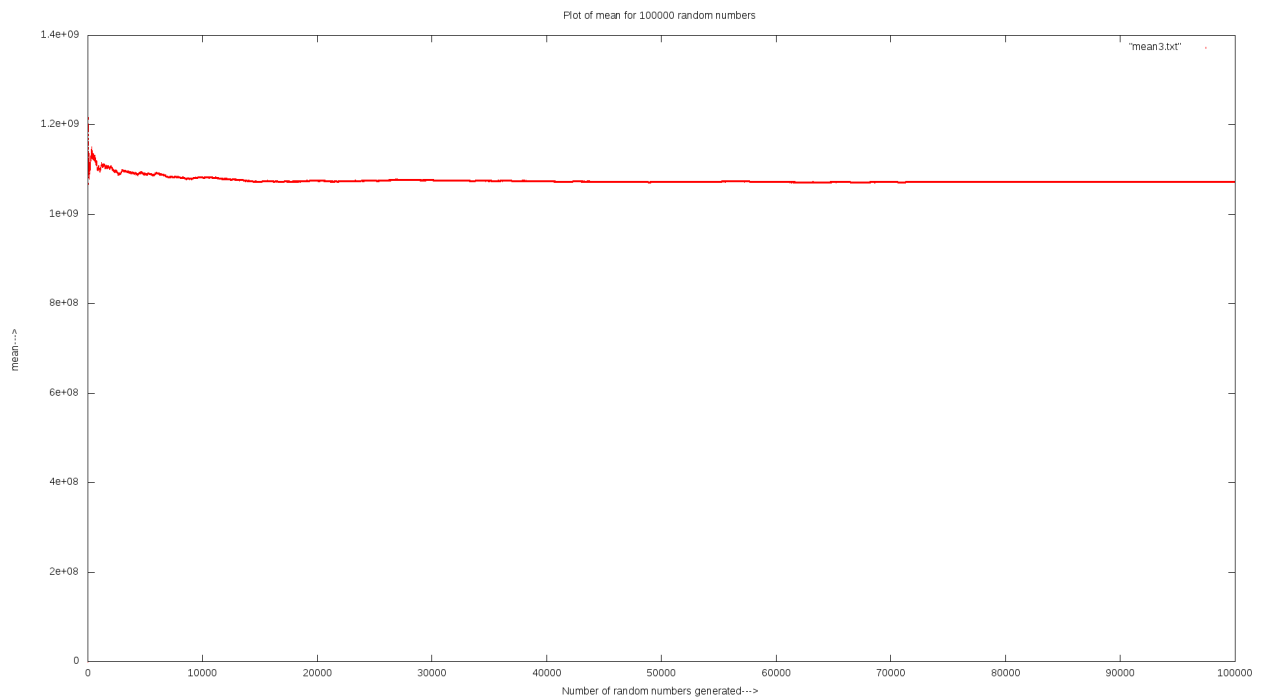


Figure 21: Mean for 100000 random numbers

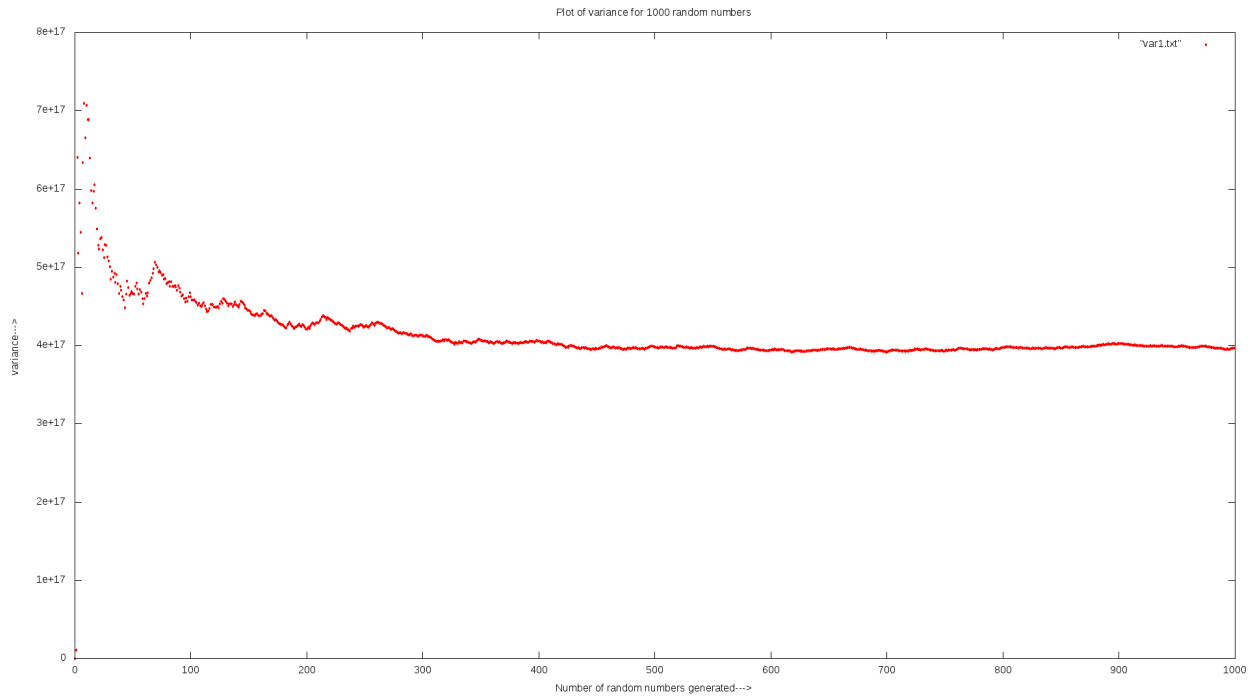


Figure 22: Variance for 1000 random numbers

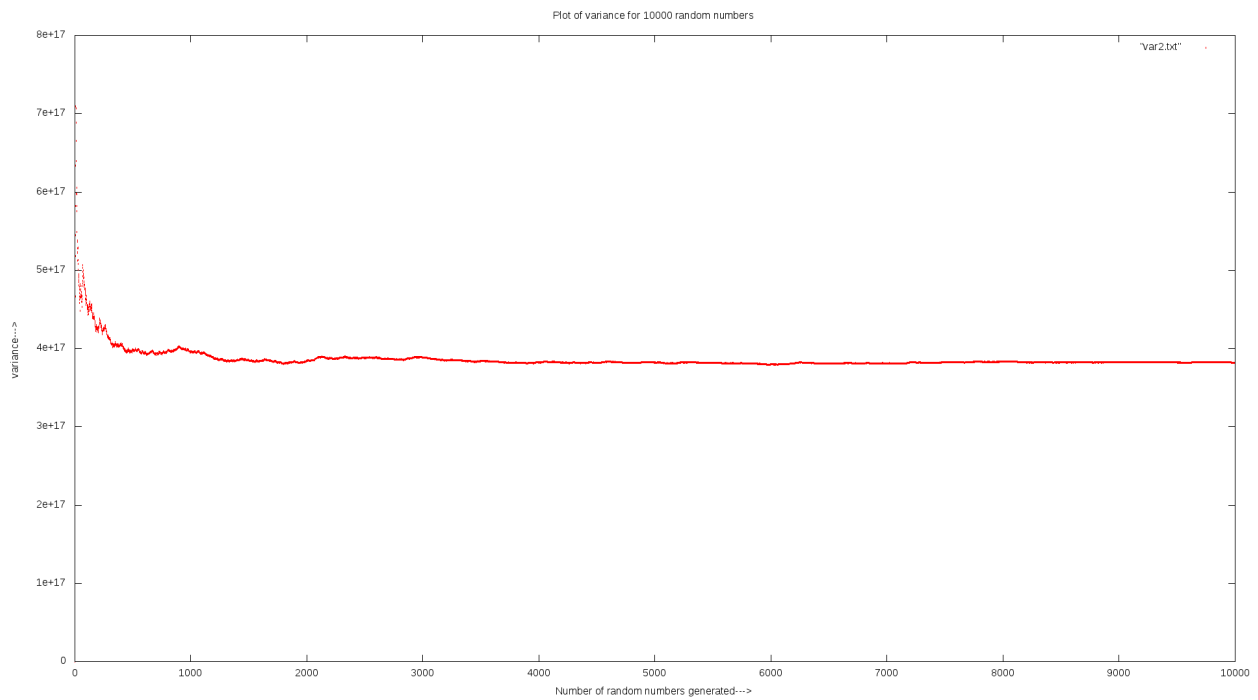


Figure 23: Variance for 10000 random numbers



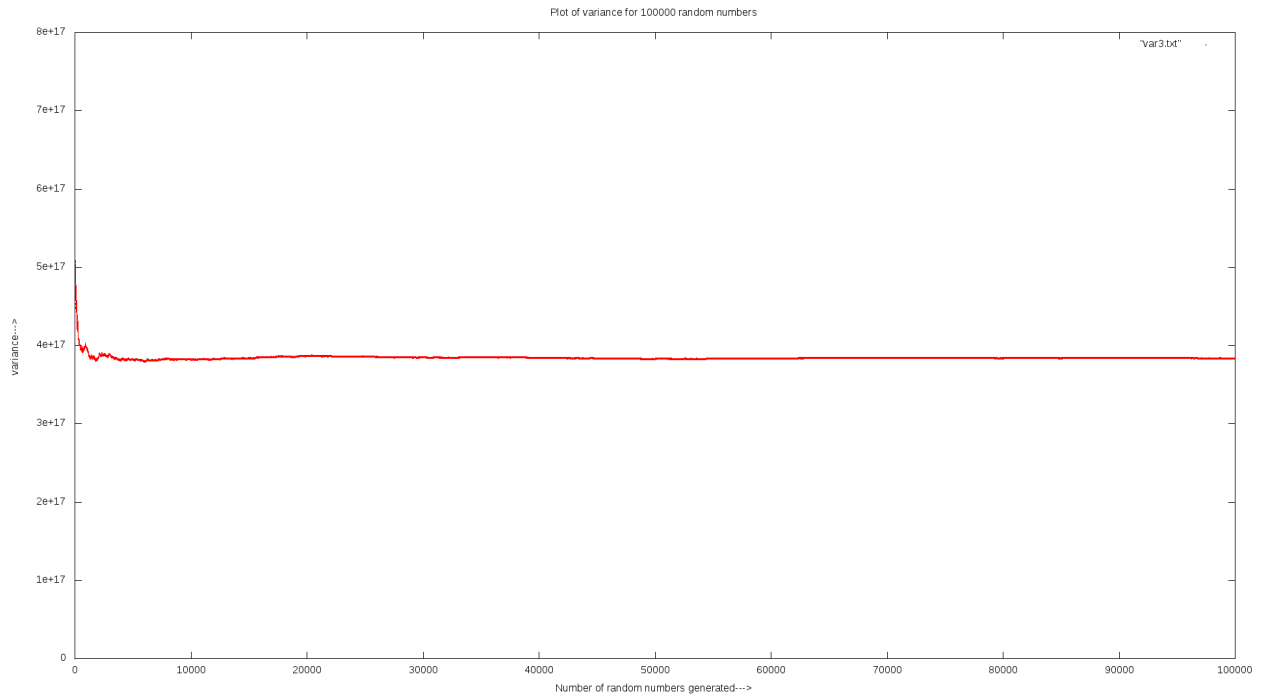


Figure 24: Variance for 100000 random numbers

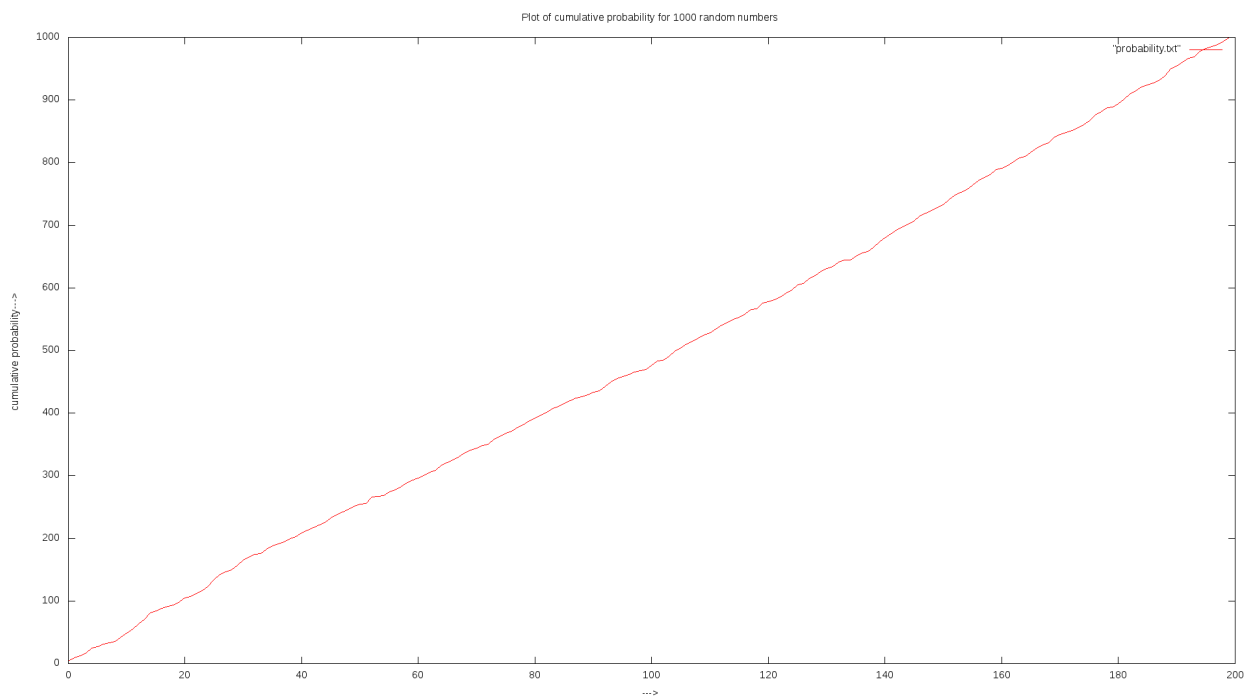


Figure 25: Plot of cumulative probability distribution for 1000 random numbers

Observations:

- The sample means and variances seem to converge to a value as observed from the graphs and the output values.
- The means converge to a value of 1072770038, and the variances converge to a value of 383718748000926080
- The probability mass function (cumulative frequencies in specific intervals) of the random numbers is observed to be approximately a straight line, parallel to the line with slope 1.
- This shows that the cumulative probability increases uniformly, resembling the probability mass function of the random variable  $\text{unif}(0,1)$ .