MA 226
Monte Carlo Simulation

Assignment - 3

(25-01-2013)

AMIT MITTAL

(Roll No.: 11012304)
Department of Mathematics
Indian Institute of Technology Guwahati

Question1:

Implement the linear congruence generator xi+1 = axi mod m to generate a sequence xi and hence uniform random numbers ui . Make use of the following set of values of a and m:

(a) a = 16807 and m = 231 − 1. (b) a = 40692 and m = 2147483399. (c) a = 40014 and m = 2147483563.

Group the values into equidistant ranges for the values of ui . 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 ?

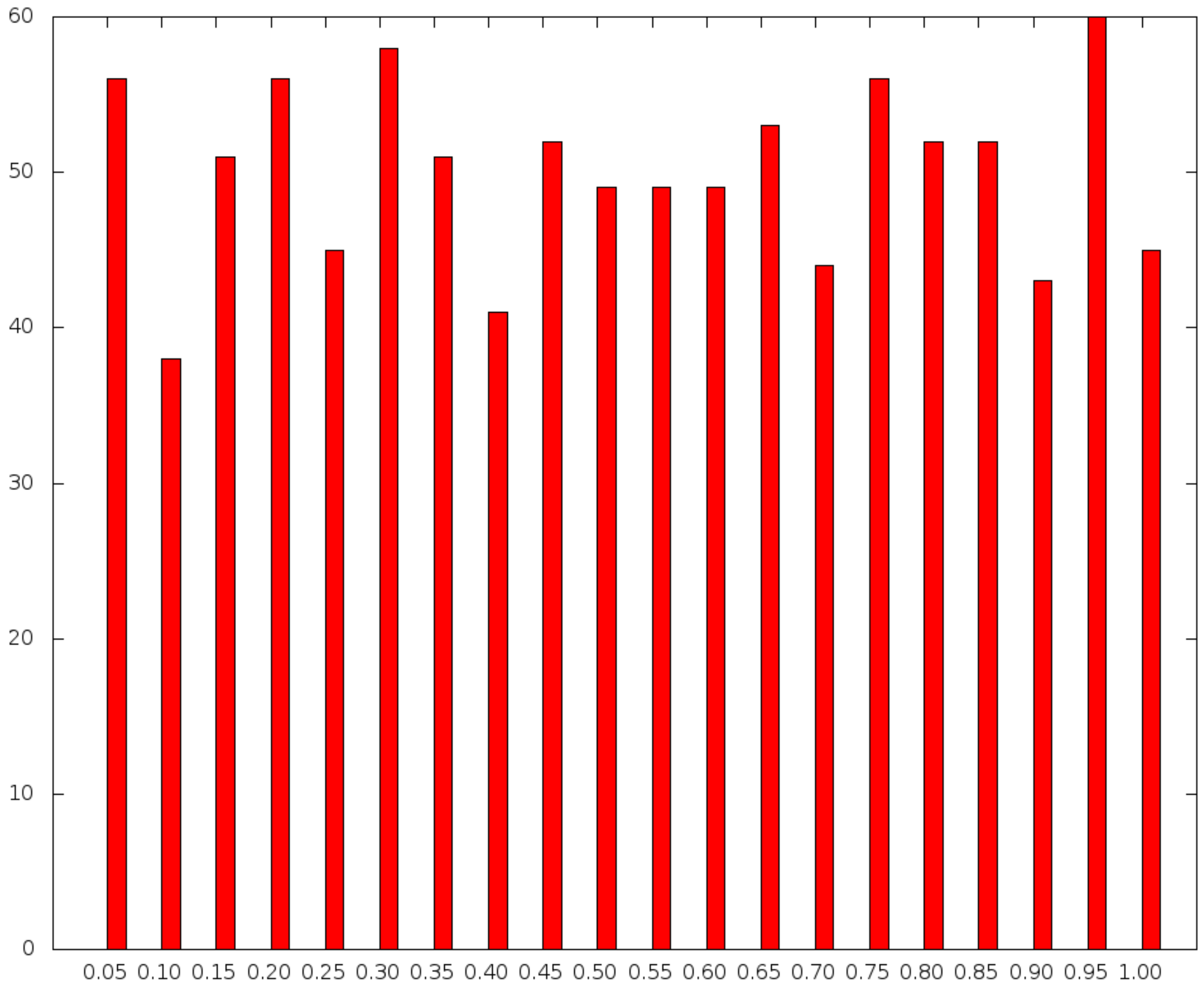
C++ code:

```
#include <iostream>
#include <cstdio>
#include <cmath>

#define LL long long

using namespace std;

double func_rand(LL int val, LL int m){
        return (double)val/m;
}

LL int func_val(LL int val, LL int a, LL int b, LL int m){
        LL next_val;
        next_val = ((a * val) + b) % m;

        return next_val;
}

void function(LL int initial, LL int a, LL int b, LL int m, int end){
        LL int val, count = 0, array[20] = {0};
        LL int index, quot;
        double random, prev;

        val = initial;
        random = func_rand(val, m);

        while(count<end){
                prev = random;
                val = func_val(val, a, b, m);
                random = func_rand(val, m);

                printf("%lf , %lf\n", prev, random);

                quot = random/0.05;
```

```
	}
		array[quot]++;
		++count;
	}

	for(index = 0 ; index<20 ; ++index){
		printf("\"%.2lf\", %lld\n", 0.05*(index+1), array[index]);
	}
}

int main(){
	LL int index, a[3] = {1000, 10000, 100000};
	LL int init_value = 3452;

	for(index = 0; index < 3; ++index){
		function(init_value, 16807, 0, pow(2, 31) - 1, a[index]);
		printf("\n");
	}

	printf("\n\n\n");

	for(index = 0; index<3; ++index){
		function(init_value, 40692, 0, 2147483399, a[index]);
		cout<<endl;
	}

	printf("\n\n\n");

	for(index = 0; index<3; ++index){
		function(init_value, 40014, 0, 2147483563, a[index]);
		cout<<endl;
	}

	printf("\n\n\n");

	return 0;
}
```

Histograms For a=16807 , m=2^31 – 1 with value of n varying

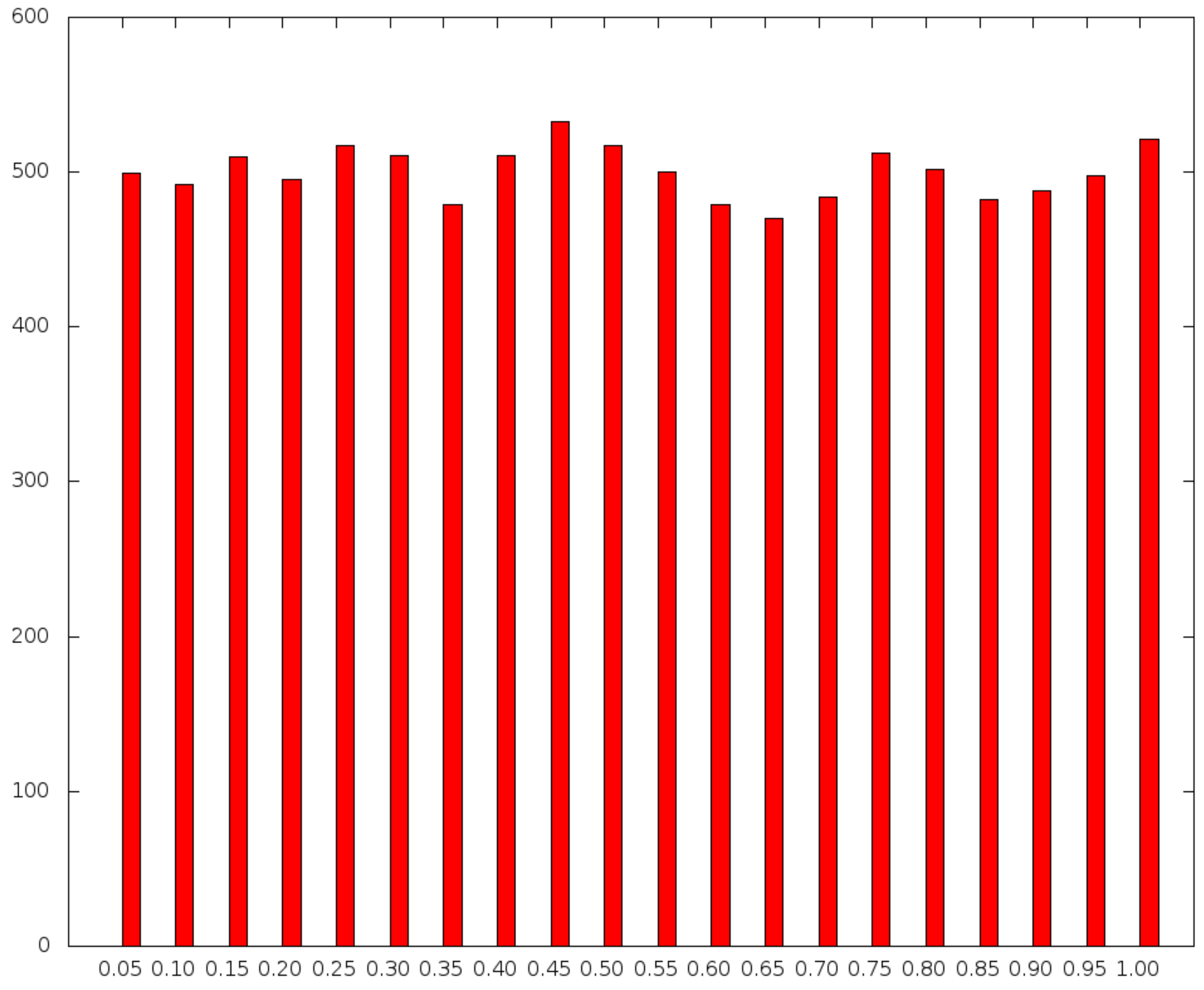
For n=1000



For a= 16807 and number of iterations = 1000 and m=2^31 - 1

Interval	Frequency
"0.00-0.05"	56
"0.05-0.10"	38
"0.10-0.15"	51
"0.15-0.20"	56
"0.20-0.25"	45
"0.25-0.30"	58
"0.30-0.35"	51
"0.35-0.40"	41
"0.40-0.45"	52
"0.45-0.50"	49
"0.50-0.55"	49
"0.55-0.60"	49
"0.60-0.65"	53
"0.65-0.70"	44
"0.70-0.75"	56
"0.75-0.80"	52
"0.80-0.85"	52
"0.85-0.90"	43
"0.90-0.95"	60
"0.95-1.00"	45

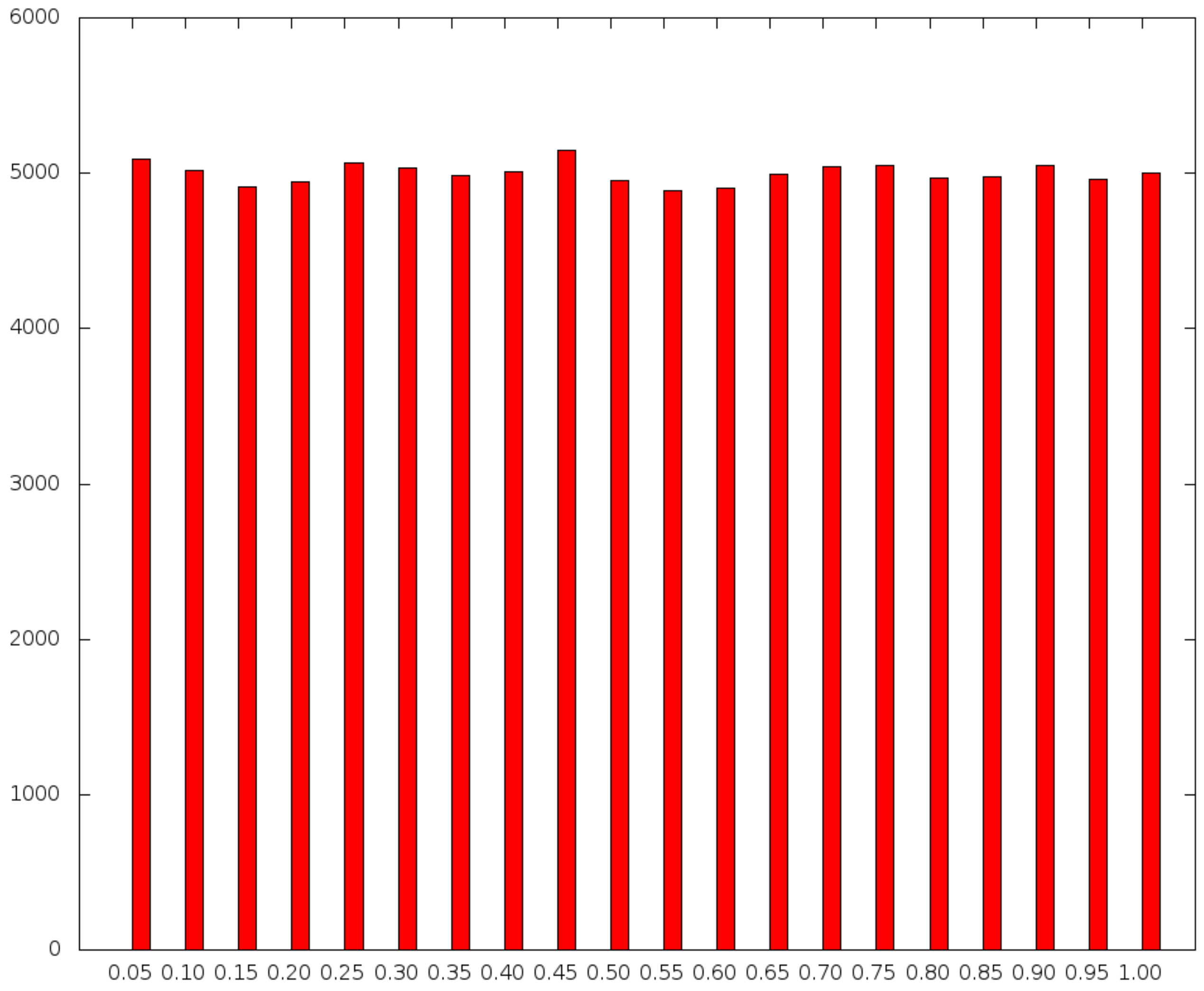
For n=10000


600
500
400
300
200
100
0
0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00

For a= 16807 and number of iterations= 10000 and m=2^31 - 1

Interval	Freq
"0.00-0.05"	499
"0.05-0.10"	492
"0.10-0.15"	510
"0.15-0.20"	495
"0.20-0.25"	517
"0.25-0.30"	511
"0.30-0.35"	479
"0.35-0.40"	511
"0.40-0.45"	533
"0.45-0.50"	517
"0.50-0.55"	500
"0.55-0.60"	479
"0.60-0.65"	470
"0.65-0.70"	484
"0.70-0.75"	512
"0.75-0.80"	502
"0.80-0.85"	482
"0.85-0.90"	488
"0.90-0.95"	498
"0.95-1.00"	521

For n=100000

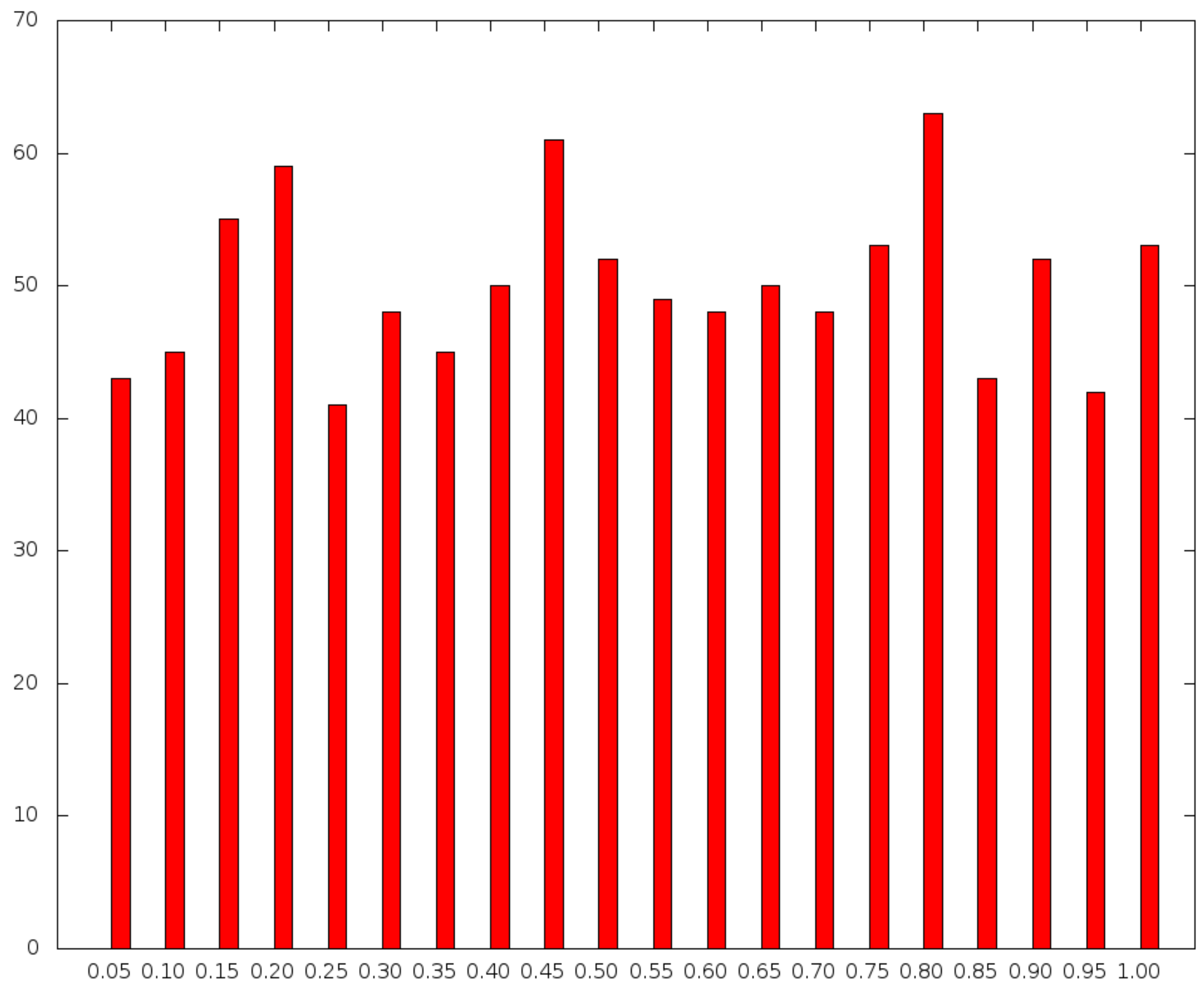

6000
5000
4000
3000
2000
1000
0
0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00

For a= 16807 and number of iterations= 100000 and m = 2^31 - 1

Interval	Freq
"0.00-0.05"	5087
"0.05-0.10"	5014
"0.10-0.15"	4912
"0.15-0.20"	4948
"0.20-0.25"	5066
"0.25-0.30"	5034
"0.30-0.35"	4987
"0.35-0.40"	5012
"0.40-0.45"	5150
"0.45-0.50"	4954
"0.50-0.55"	4888
"0.55-0.60"	4900
"0.60-0.65"	4996
"0.65-0.70"	5044
"0.70-0.75"	5049
"0.75-0.80"	4972
"0.80-0.85"	4978
"0.85-0.90"	5048
"0.90-0.95"	4958
"0.95-1.00"	5003

Histograms for a= 40692 , m=2147483399 with value of n varying

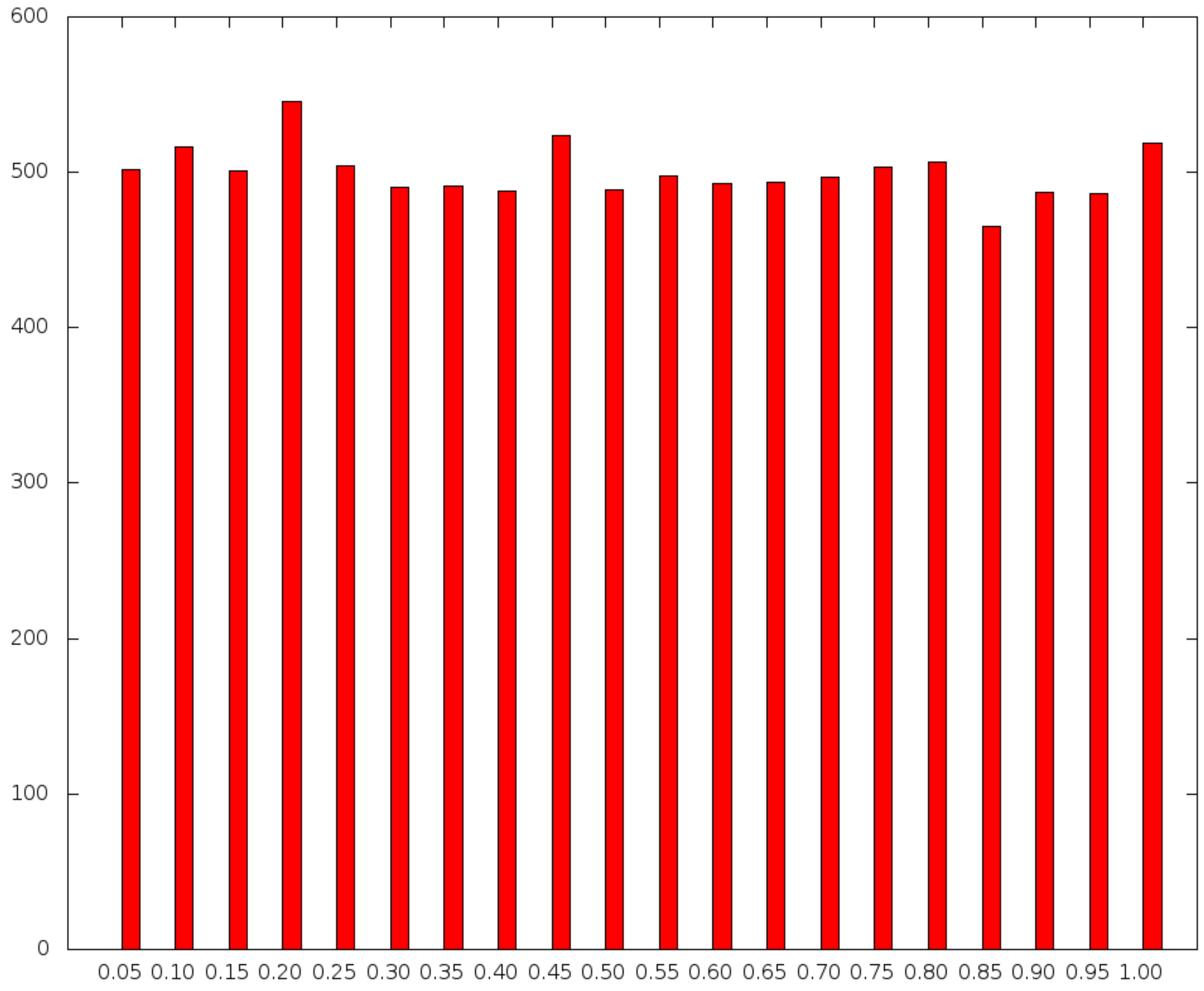
For n=1000



<u>For a= 40692 and number of iterations = 1000 and m=2147483399</u>

Interval	Freq
"0.00-0.05"	43
"0.05-0.10"	45
"0.10-0.15"	55
"0.15-0.20"	59
"0.20-0.25"	41
"0.25-0.30"	48
"0.30-0.35"	45
"0.35-0.40"	50
"0.40-0.45"	61
"0.45-0.50"	52
"0.50-0.55"	49
"0.55-0.60"	48
"0.60-0.65"	50
"0.65-0.70"	48
"0.70-0.75"	53
"0.75-0.80"	63
"0.80-0.85"	43
"0.85-0.90"	52
"0.90-0.95"	42
"0.95-1.00"	53

For n=10000

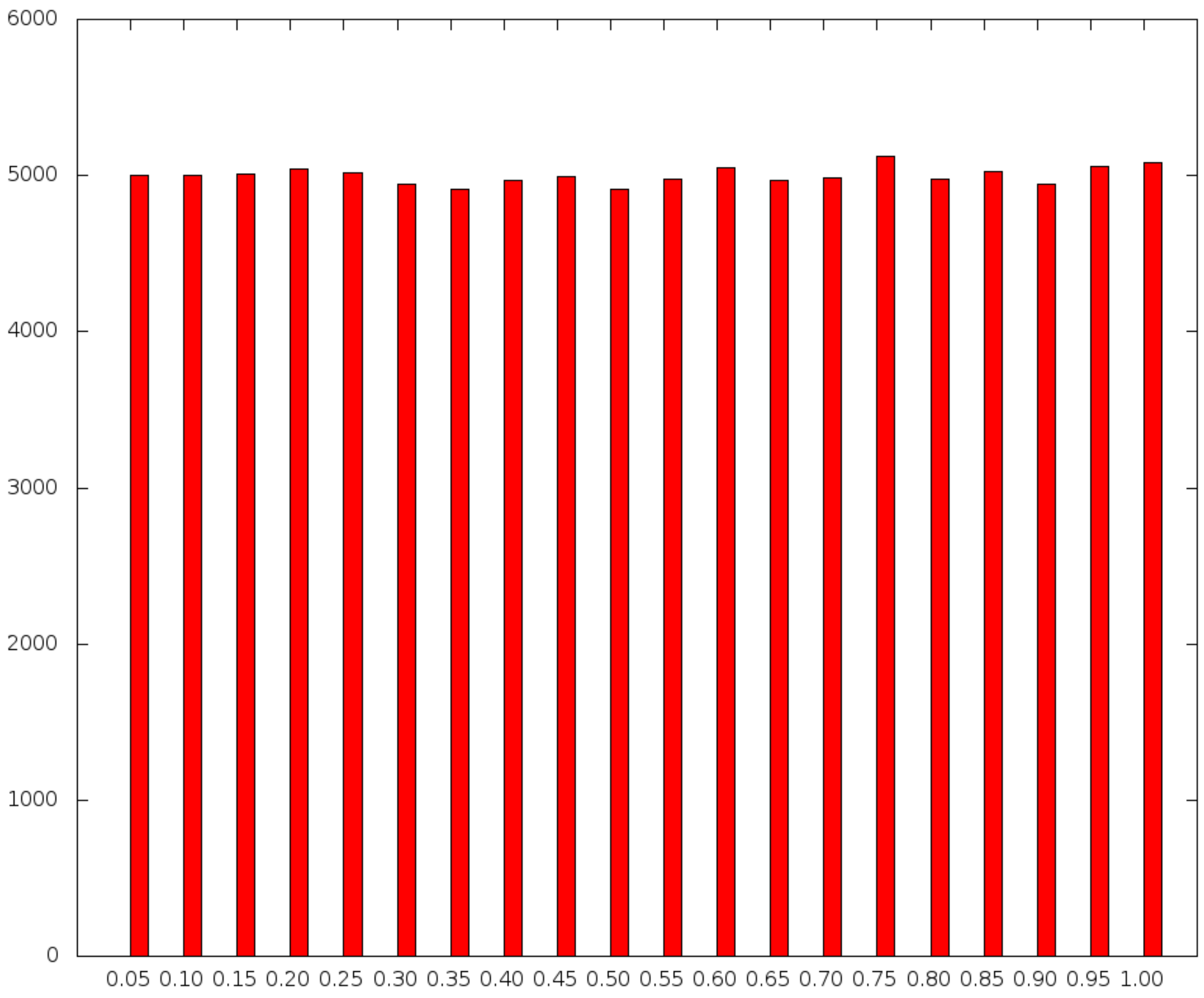


0
100
200
300
400
500
600
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00

For a= 40692 and number of iterations= 10000 and m=2147483399

Interval	Freq
"0.00-0.05"	502
"0.05-0.10"	516
"0.10-0.15"	501
"0.15-0.20"	546
"0.20-0.25"	504
"0.25-0.30"	490
"0.30-0.35"	491
"0.35-0.40"	488
"0.40-0.45"	524
"0.45-0.50"	489
"0.50-0.55"	498
"0.55-0.60"	493
"0.60-0.65"	494
"0.65-0.70"	497
"0.70-0.75"	503
"0.75-0.80"	507
"0.80-0.85"	465
"0.85-0.90"	487
"0.90-0.95"	486
"0.95-1.00"	519

For n=100000



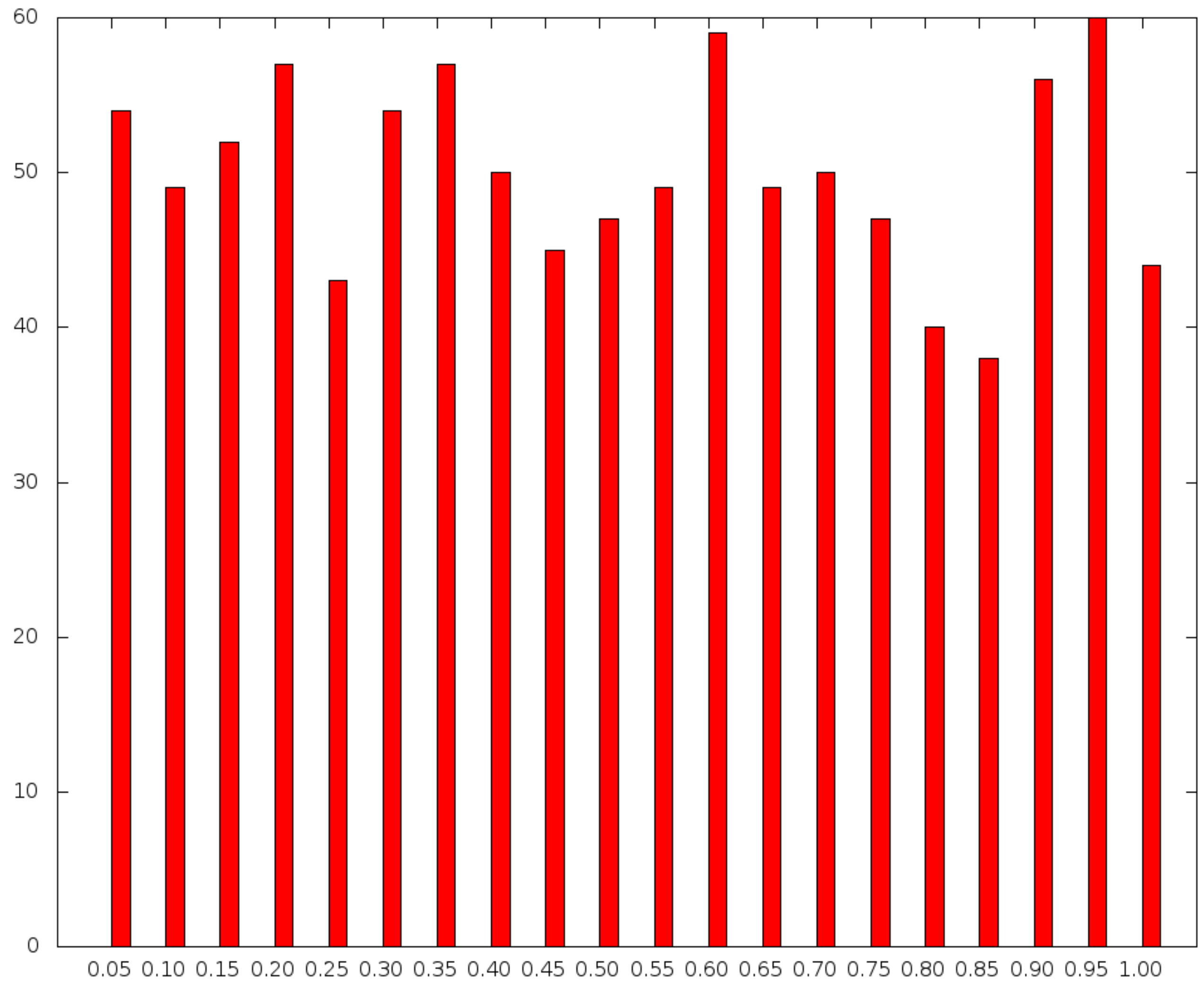
6000
5000
4000
3000
2000
1000
0
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00

For a= 40692 and number of iterations= 100000 and m=2147483399

Interval	**Freq**
"0.00-0.05"	5002
"0.05-0.10"	5005
"0.10-0.15"	5007
"0.15-0.20"	5044
"0.20-0.25"	5014
"0.25-0.30"	4944
"0.30-0.35"	4913
"0.35-0.40"	4971
"0.40-0.45"	4991
"0.45-0.50"	4910
"0.50-0.55"	4977
"0.55-0.60"	5052
"0.60-0.65"	4970
"0.65-0.70"	4989
"0.70-0.75"	5126
"0.75-0.80"	4979
"0.80-0.85"	5023
"0.85-0.90"	4941
"0.90-0.95"	5056
"0.95-1.00"	5086

Histograms For a= 40014 , m=2147483563 and value of n varying

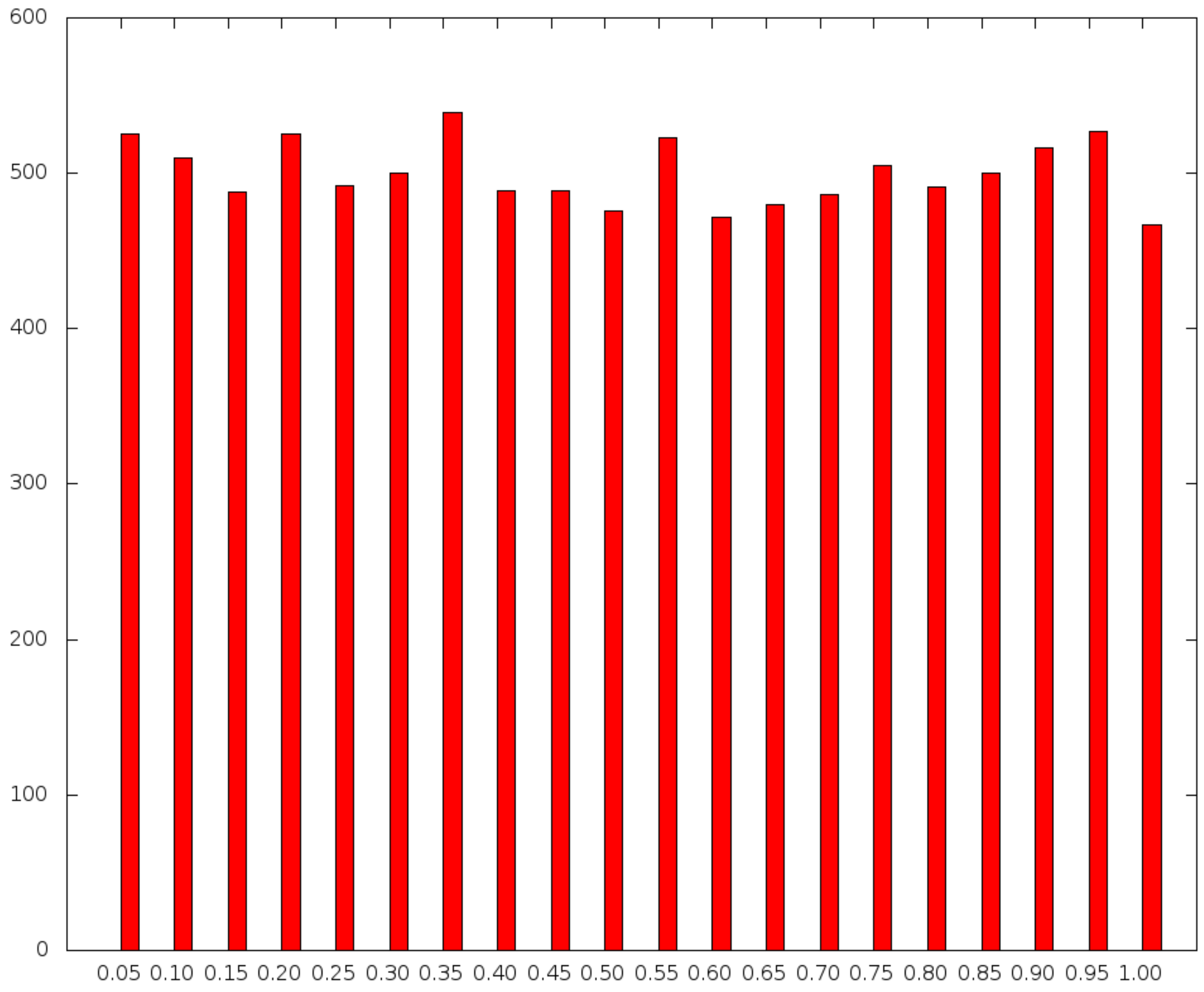
For n=1000



For a= 40014 and number of iterations= 1000 and m=2147483563

Interval	Freq
"0.00-0.05"	54
"0.05-0.10"	49
"0.10-0.15"	52
"0.15-0.20"	57
"0.20-0.25"	43
"0.25-0.30"	54
"0.30-0.35"	57
"0.35-0.40"	50
"0.40-0.45"	45
"0.45-0.50"	47
"0.50-0.55"	49
"0.55-0.60"	59
"0.60-0.65"	49
"0.65-0.70"	50
"0.70-0.75"	47
"0.75-0.80"	40
"0.80-0.85"	38
"0.85-0.90"	56
"0.90-0.95"	60
"0.95-1.00"	44

For n=10000

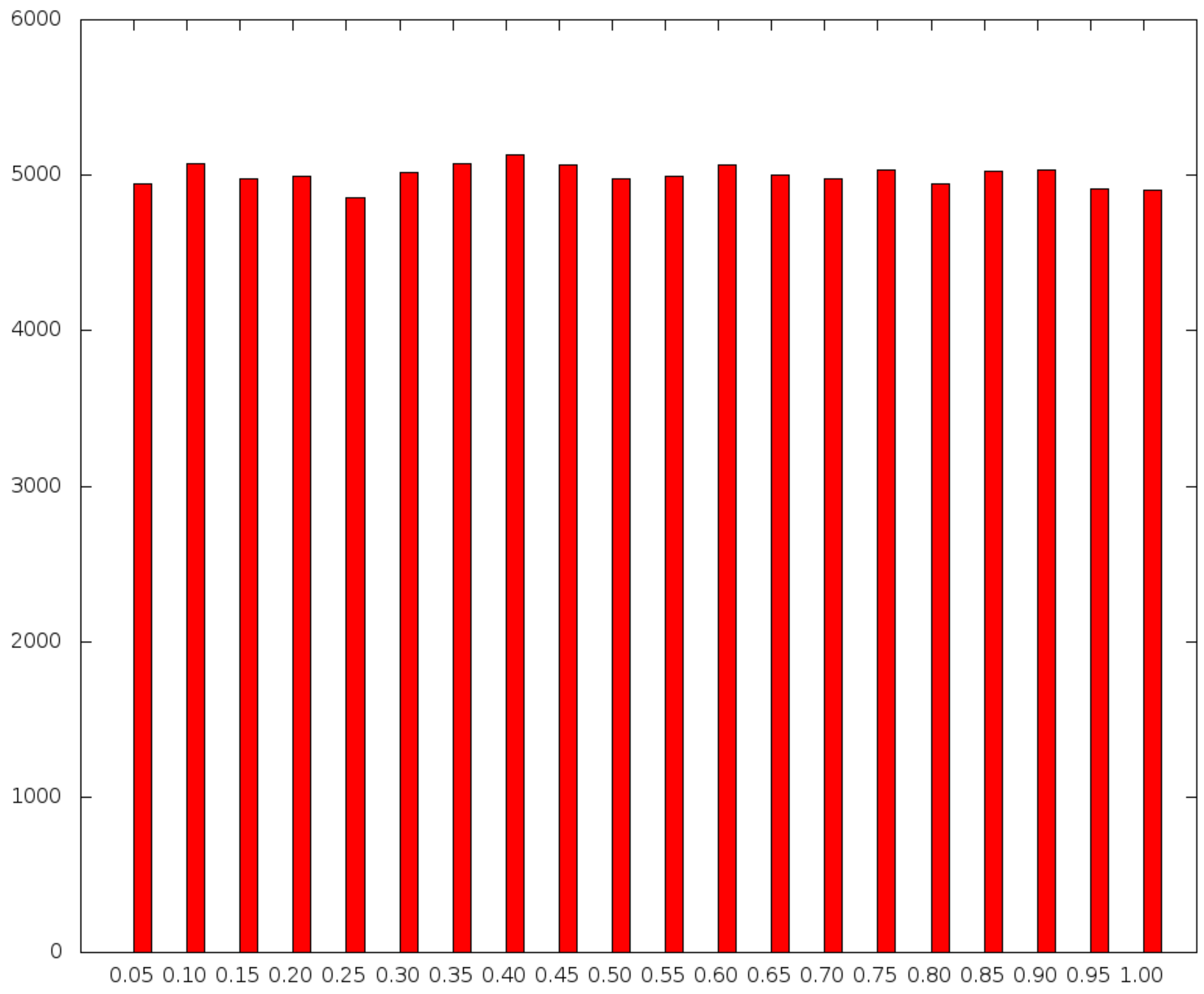


600
500
400
300
200
100
0
0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00

For a= 40014 and number of iterations= 10000 and m=2147483563

Interval	Freq
"0.00-0.05"	525
"0.05-0.10"	510
"0.10-0.15"	488
"0.15-0.20"	525
"0.20-0.25"	492
"0.25-0.30"	500
"0.30-0.35"	539
"0.35-0.40"	489
"0.40-0.45"	489
"0.45-0.50"	476
"0.50-0.55"	523
"0.55-0.60"	472
"0.60-0.65"	480
"0.65-0.70"	486
"0.70-0.75"	505
"0.75-0.80"	491
"0.80-0.85"	500
"0.85-0.90"	516
"0.90-0.95"	527
"0.95-1.00"	467

For n=100000

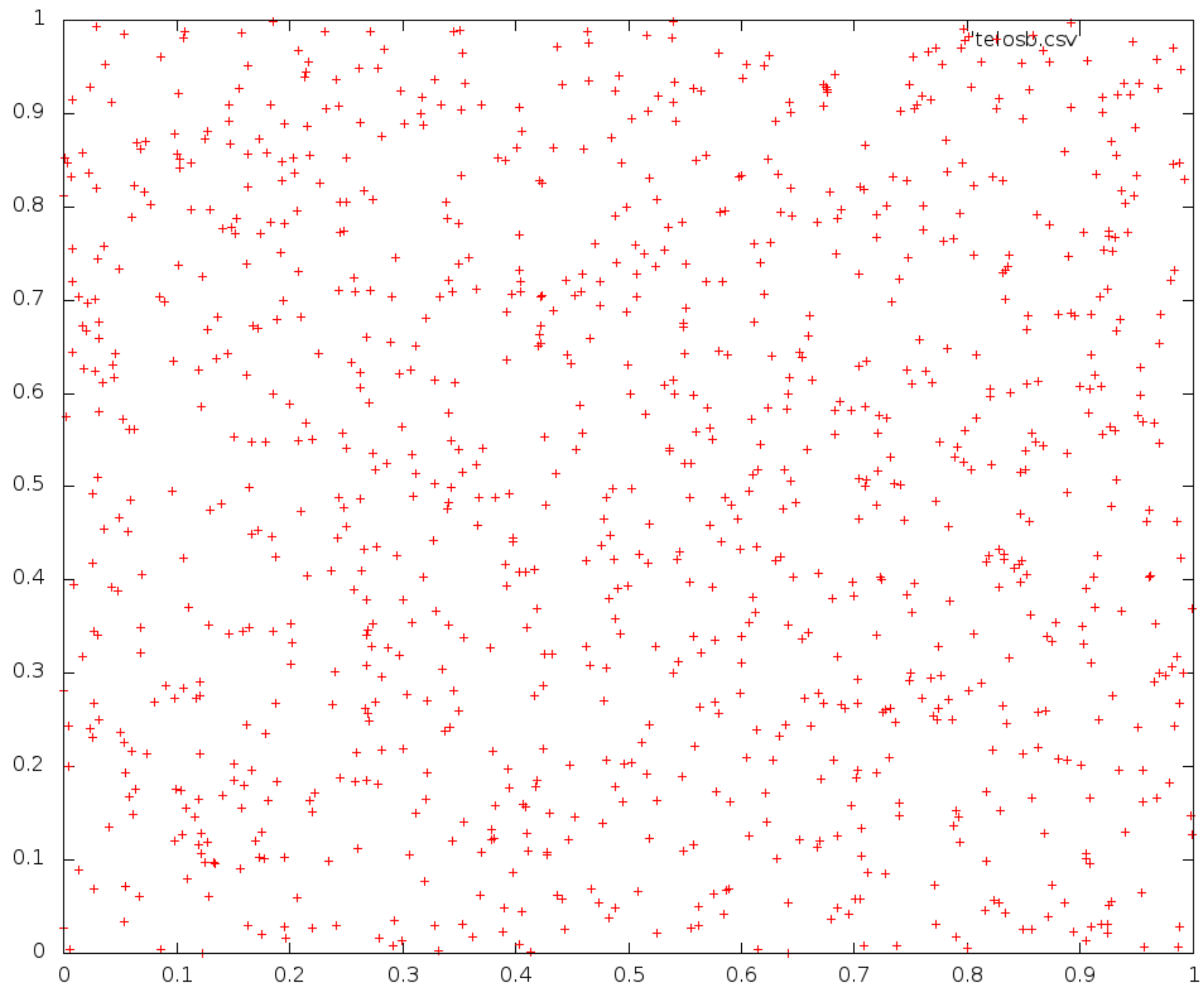


6000
5000
4000
3000
2000
1000
0
0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00

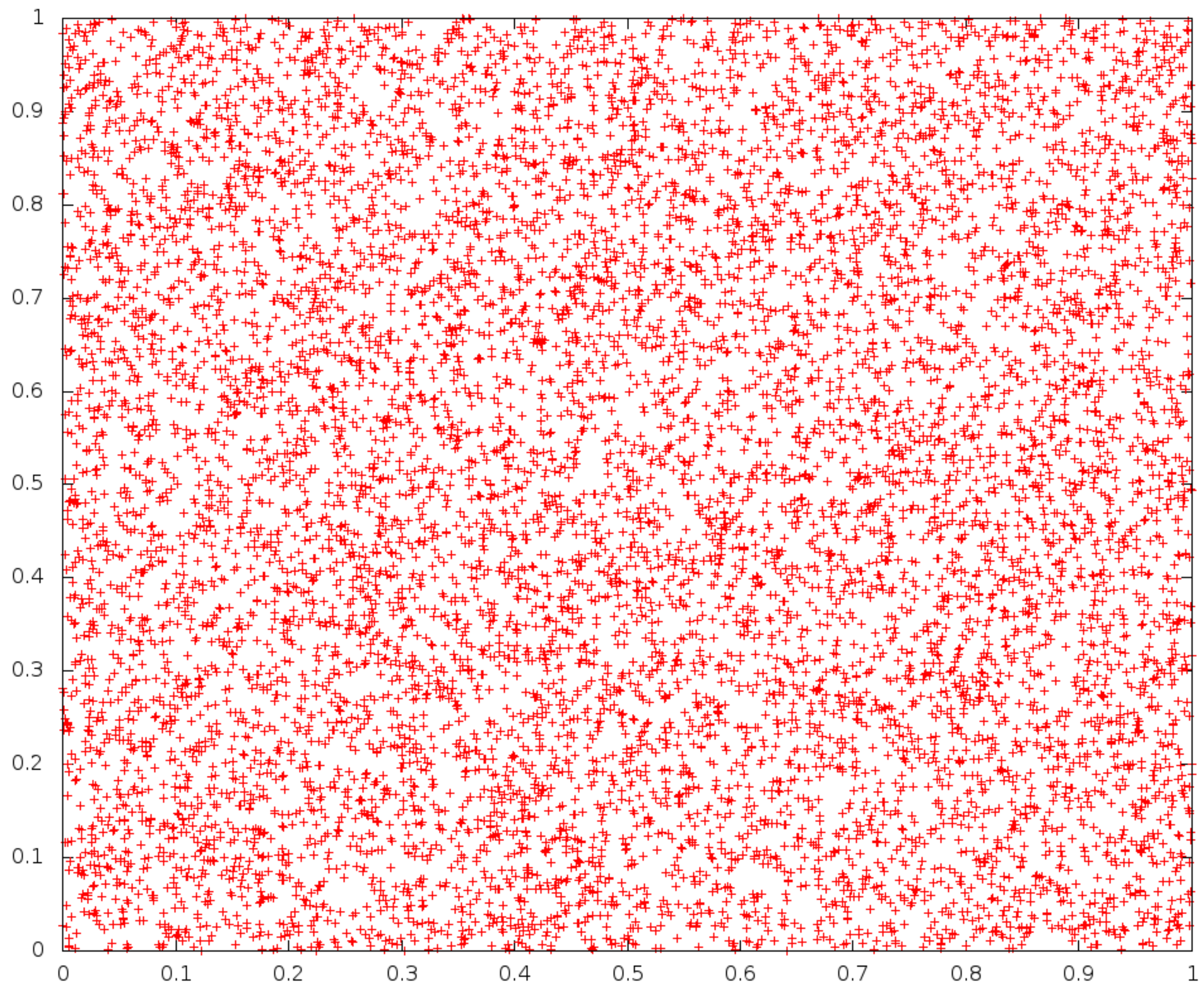
For a= 40014 and number of iterations= 100000 and m=2147483563

Interval	Freq
"0.00-0.05"	4946
"0.05-0.1"	5073
"0.1-0.15"	4974
"0.15-0.20"	4991
"0.20-0.25"	4855
"0.25-0.30"	5017
"0.30-0.35"	5073
"0.35-0.40"	5128
"0.40-0.45"	5070
"0.45-0.50"	4979
"0.50-0.55"	4997
"0.55-0.60"	5064
"0.60-0.65"	5001
"0.65-0.70"	4973
"0.70-0.75"	5035
"0.75-0.80"	4943
"0.80-0.85"	5023
"0.85-0.90"	5036
"0.90-0.95"	4916
"0.95-1.00"	4906

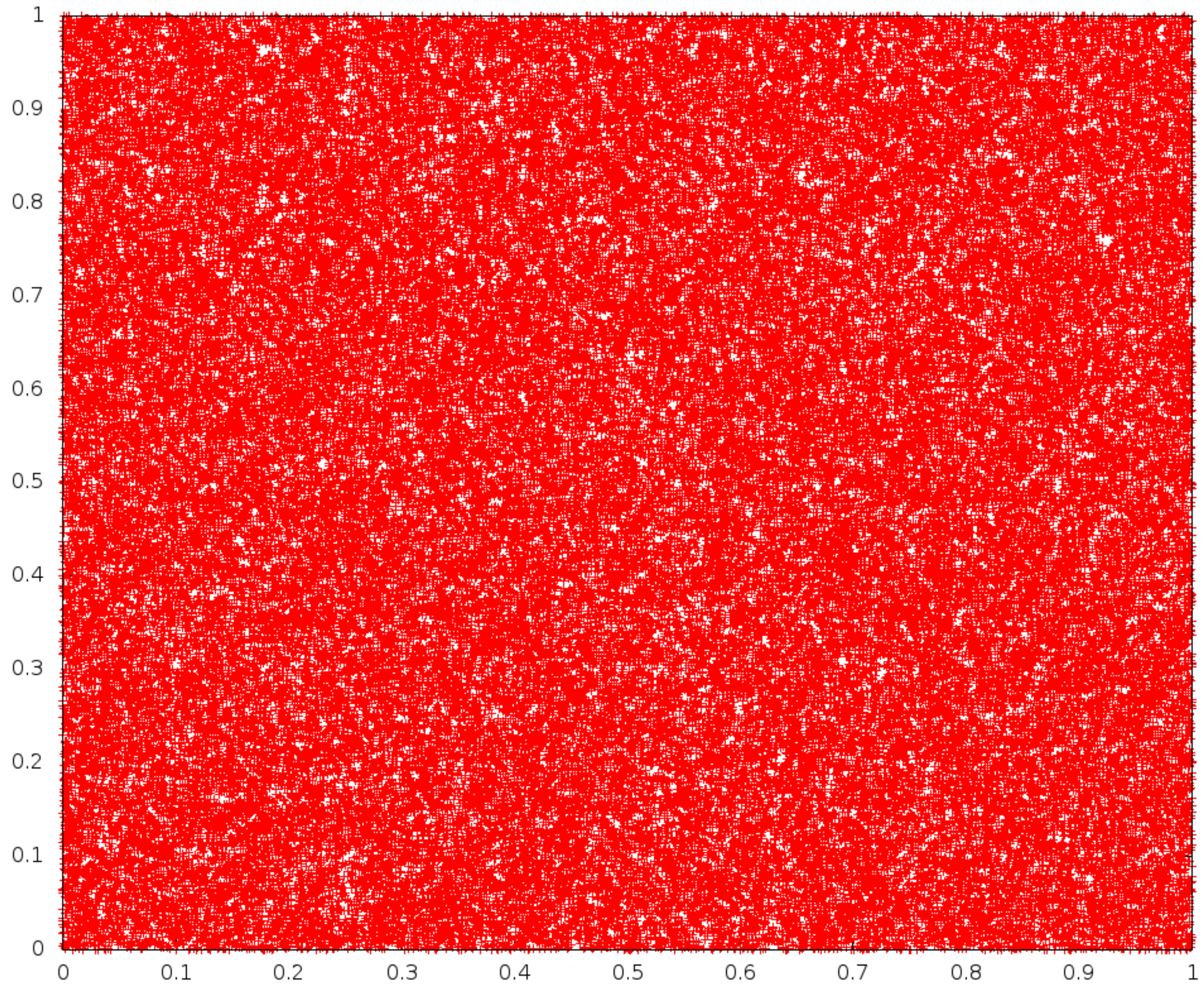
Plot of (u_i, u_{i+1}) on a unit square for a= 16807 , m= 2^31-1 and n=1000



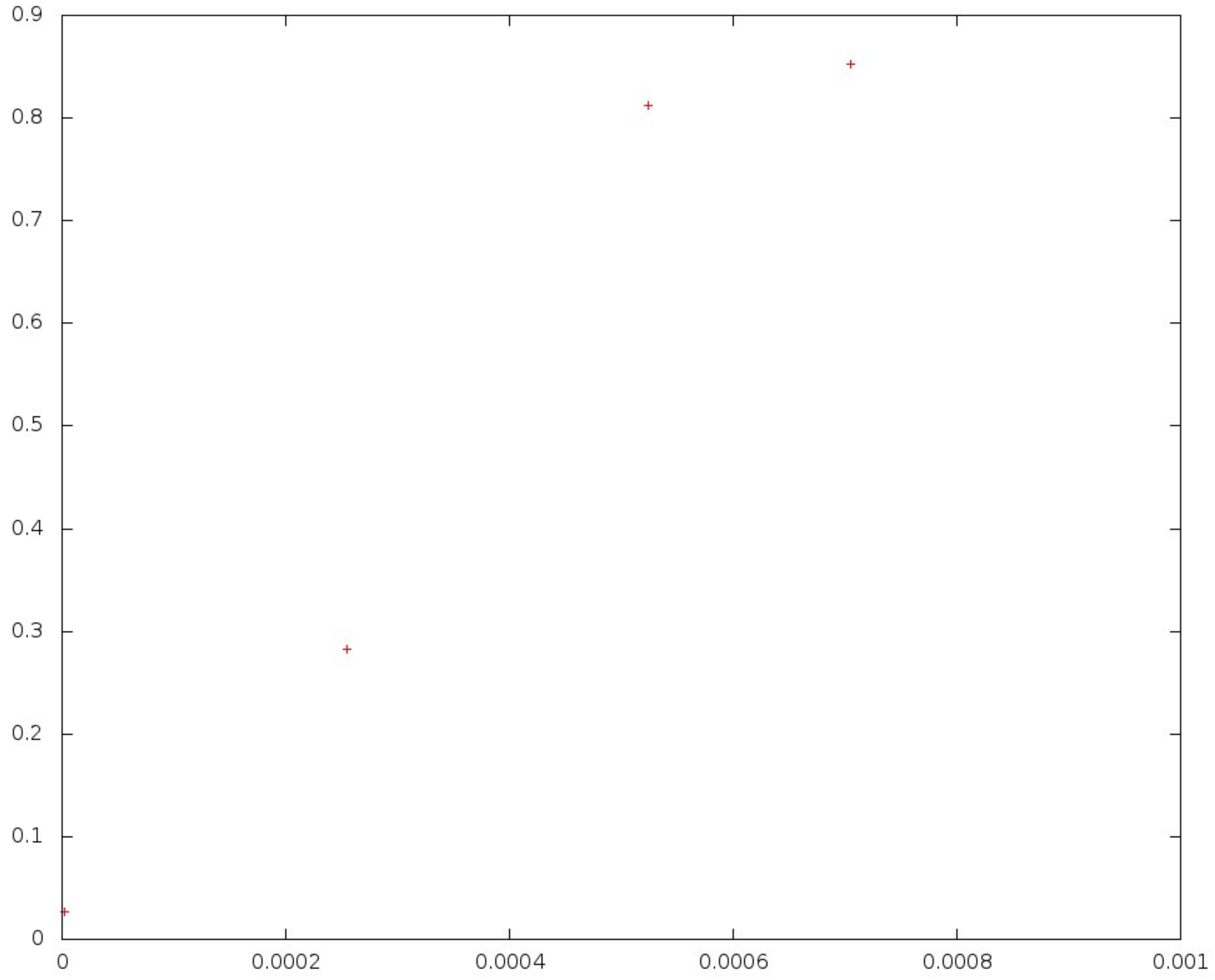
Plot of ($u_{i,}$, u_{i+1}) on a unit square for a= 16807,m= 2^31-1 and n= 10000



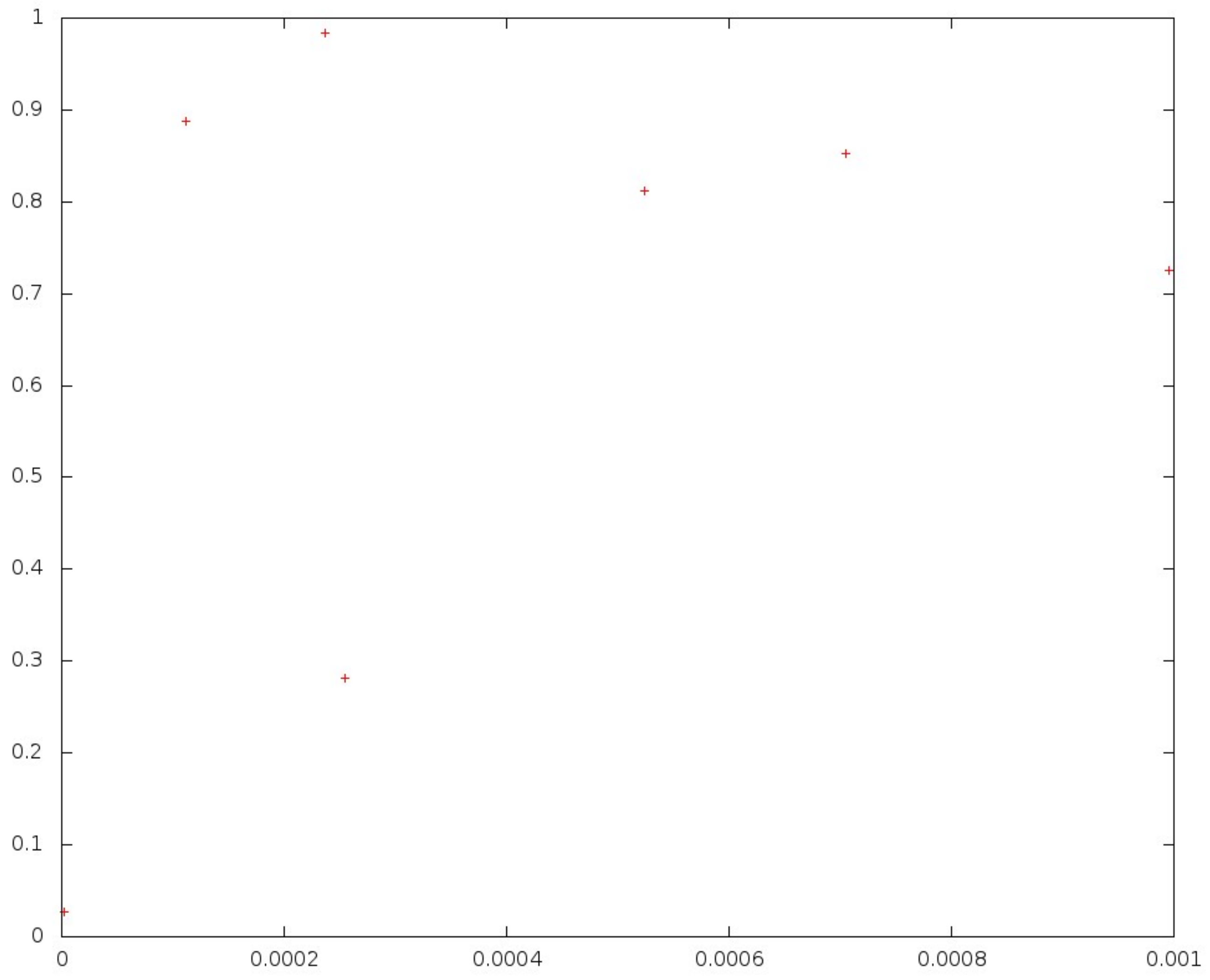
Plot of (u_{i}, u_{i+1}) on a unit square for a= 16807,m= 2^31-1 and n= 100000



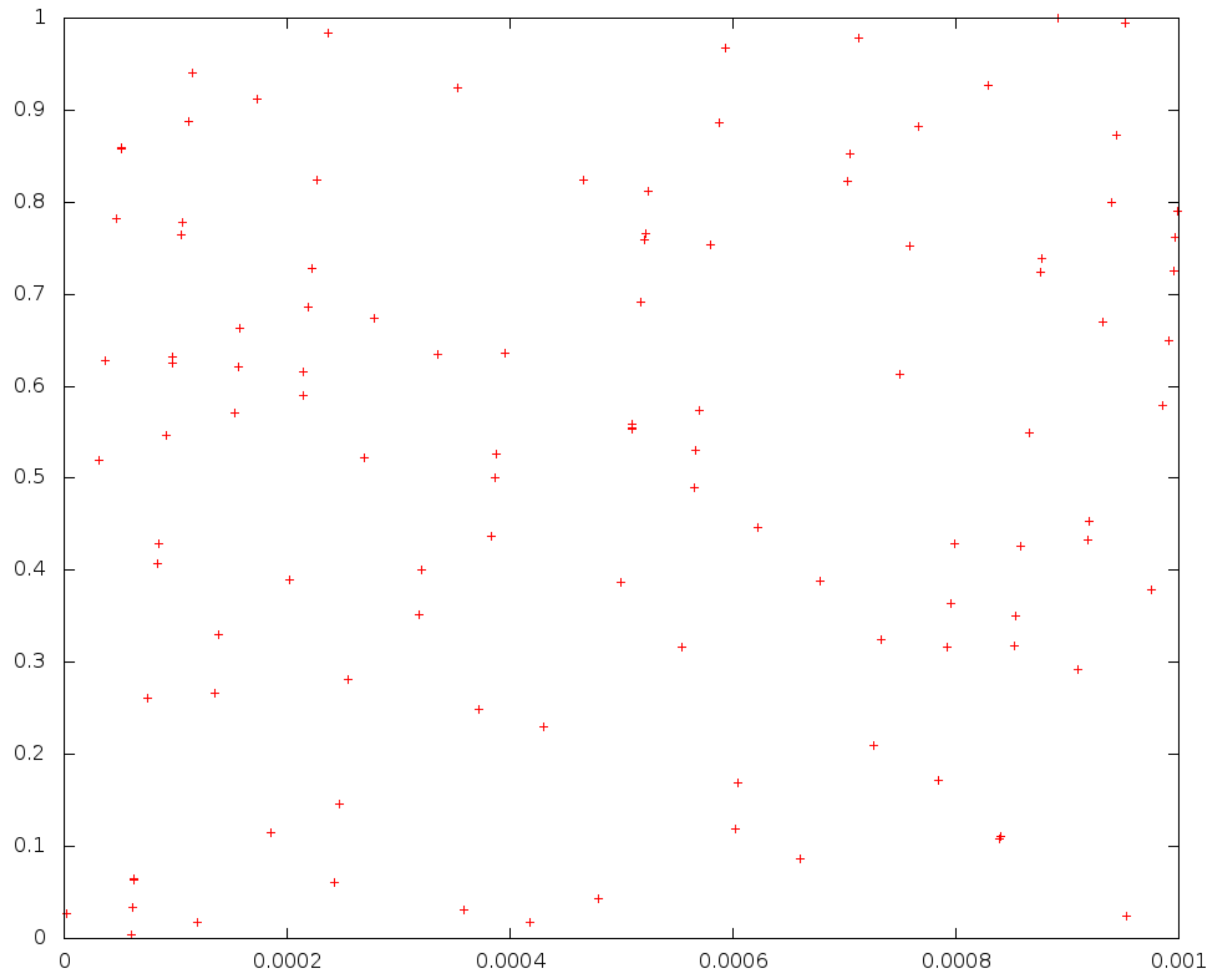
Plot of (u_i , u_{i+1}) on a unit square for a= 16807 , m= 2^31-1 and n=1000 zoomed to u_i < 0.001 and yrange between 0 and 1



Plot of (u_{i}, u_{i+1}) on a unit square for a= 16807 , m= 2^31-1 and n=10000 zoomed to u_i < 0.001 and yrange between 0 and 1



Plot of (u_i, u_{i+1}) on a unit square for a= 16807 , m= 2^31-1 and n=100000 zoomed to u_i < 0.001 and yrange between 0 and 1



OBSERVATIONS

1. **As the value of 'n' (The number of iterations) increases,the values of u_i converge towards a uniform distribution.**
2. **On plotting the graph of ($u_{i,}$ $u_{i+1)}$ we observe that the random numbers generated are uniformly concentrated in the range [0,1].Also,the density of these random numbers increases as the number of iterations increase.**
3. **On zooming into the given interval of [0,0.001],we observe that the density of random numbers generated decreases significantly. However,the density increases as the number of iterations increase.**

RESULT

1.Since,the values of u_i generated are uniformly distributed in the interval [0,1],we conclude that the given random value generator is an efficient generator.
2.Also,as the number of iterations increases,the density of the random values in a particular interval increases. This observation strengthens our conclusion about the efficiency of the given generator.

Question2:

Consider the extended Fibonacci generator :
$U_i = (U_{i-17} + U_{i-5}) \bmod 231$.
(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.

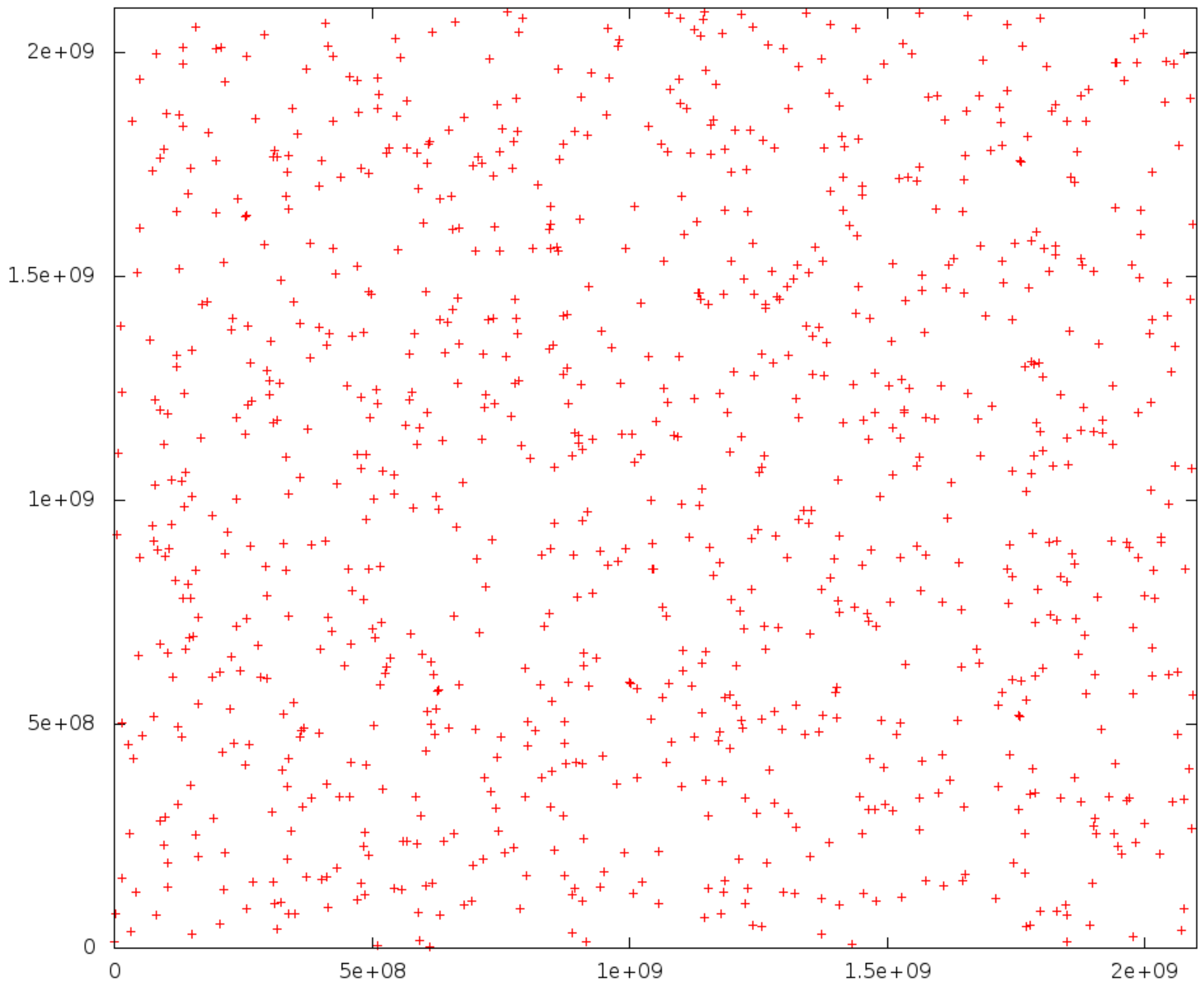
C++ code:

```
#include <iostream>
#include <cstdio>
#include <cmath>

#define LL long long
#define FIB 17
#define INTERVAL 100000

using namespace std;

double func_rand(LL int val, LL int m){
        return (double)val/m;
}

LL int func_val(LL int val, LL int a, LL int b, LL int m){
        LL next_val;
        next_val = ((a * val) + b) % m;

        return next_val;
}

void probab_distribution(LL int array[], int end){
        LL int pro[100000] = {0}, index, rem, max=-1;

        for(index = 0; index<end; ++index){
                rem = array[index]/INTERVAL;
                if(rem>max)
                        max = rem;
                ++pro[rem];
        }

        for(index = 1;index<=max; ++index){
                pro[index] += pro[index-1];
        }
```

```
	for(index = 0; index<=max; ++index){
		printf("%lld %lf\n",(index+1)*INTERVAL, (double)pro[index]/end);
	}
}

void function(LL int initial, LL int a, LL int b, LL int m, int end){
	LL int val, count = 0;
	LL int index, quot, array[110000], prev, sum;
	double mean, variance, sum_var, sum_auto, autocorr;

	val = initial;

	for(count = 0; count<FIB; ++count){
		prev = val;
		val = func_val(val, a, b, m);

		printf("%lld , %lld\n", prev, val);
		array[count] = val;
	}

	while(count<end){
		prev = val;
		val = array[count-17] + array[count-5];
		val = val % (LL int)pow(2,31);
		array[count] = val;

		printf("%lld , %lld\n", prev, val);
		++count;
	}

	sum = 0;
	for(index = 0; index<end; ++index){
		sum+=array[index];
	}
	mean = (double)sum/end;

	sum_var = 0.0;
	for(index = 0; index<end; ++index){
		sum_var += pow((double)(array[index] - mean), 2);
	}
	variance = (double)sum_var/end;

	printf("\nMean=%lf Variance=%lf\n", mean, variance);

	if(end == 1000){
		int lag;
		sum_auto = 0.0;
		for(lag = 1;lag<6;++lag){
			for(index = lag; index<end; ++index){
				sum_auto +=(double)((double)(array[index] - mean)*(double)
(array[index-lag] - mean));
			}
			autocorr = (double)sum_auto/sum_var;

			printf("\nAutocorrelation %d = %lf\n", lag, autocorr);
		}
	}

	if(end == 1000){
		probab_distribution(array, end);
	}
}

int main(){
	LL int index, a[3] = {1000, 10000, 100000};
	LL int init_value = 3452;

	for(index = 0; index<3; ++index){
		function(init_value, 4179, 214, 2147483563, a[index]);
		cout<<endl;
	}

	return 0;
}
```

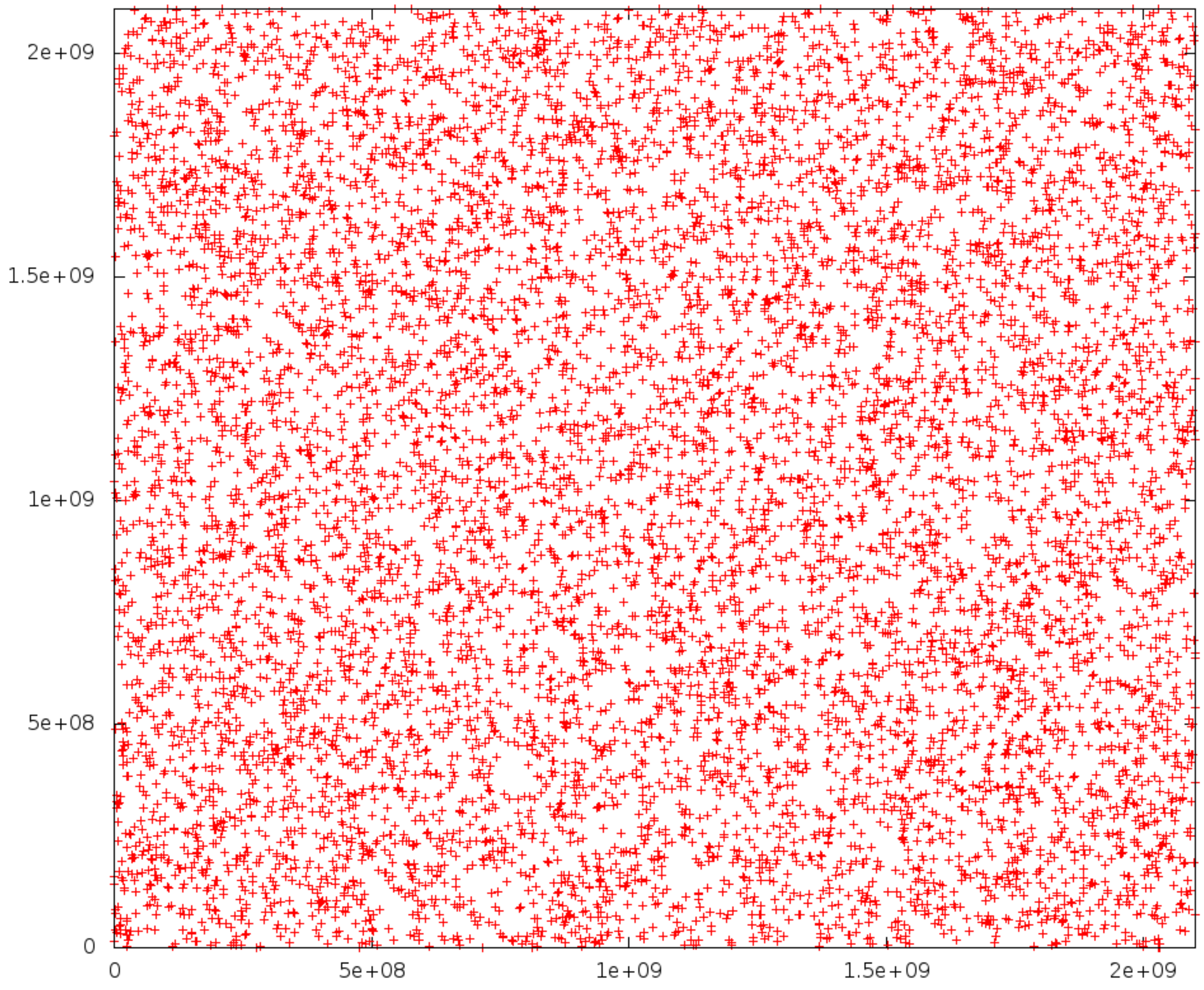
Plot of (u_{i}, u_{i+1}) for a constant value of 'a' and 'm' with n=1000

a = 4179 , b = 214 , m = 2147483563



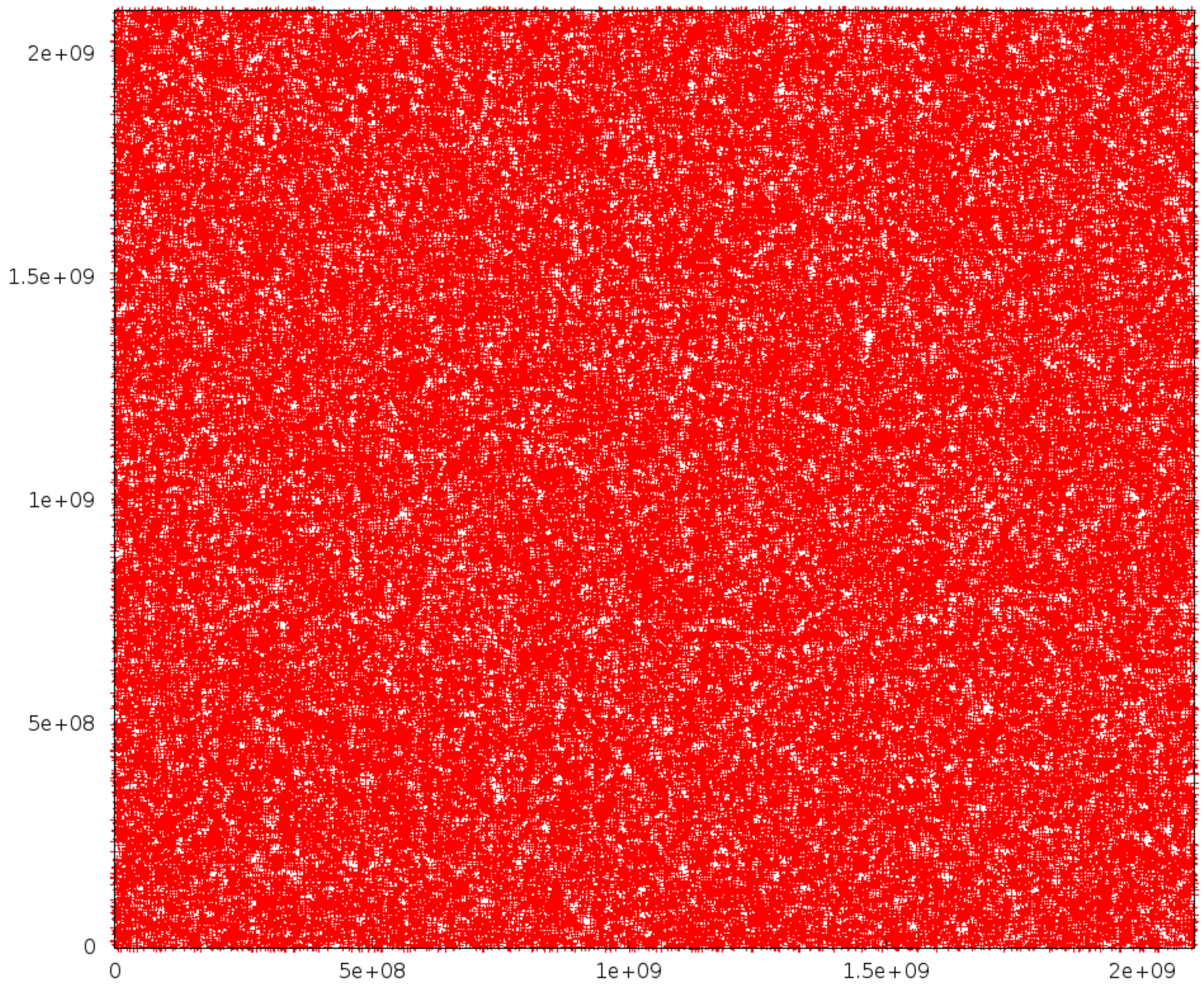
Plot of (u_i, u_{i+1}) for a constant value of 'a' and 'm' with n=10000

a = 4179 , b = 214 , m = 2147483563

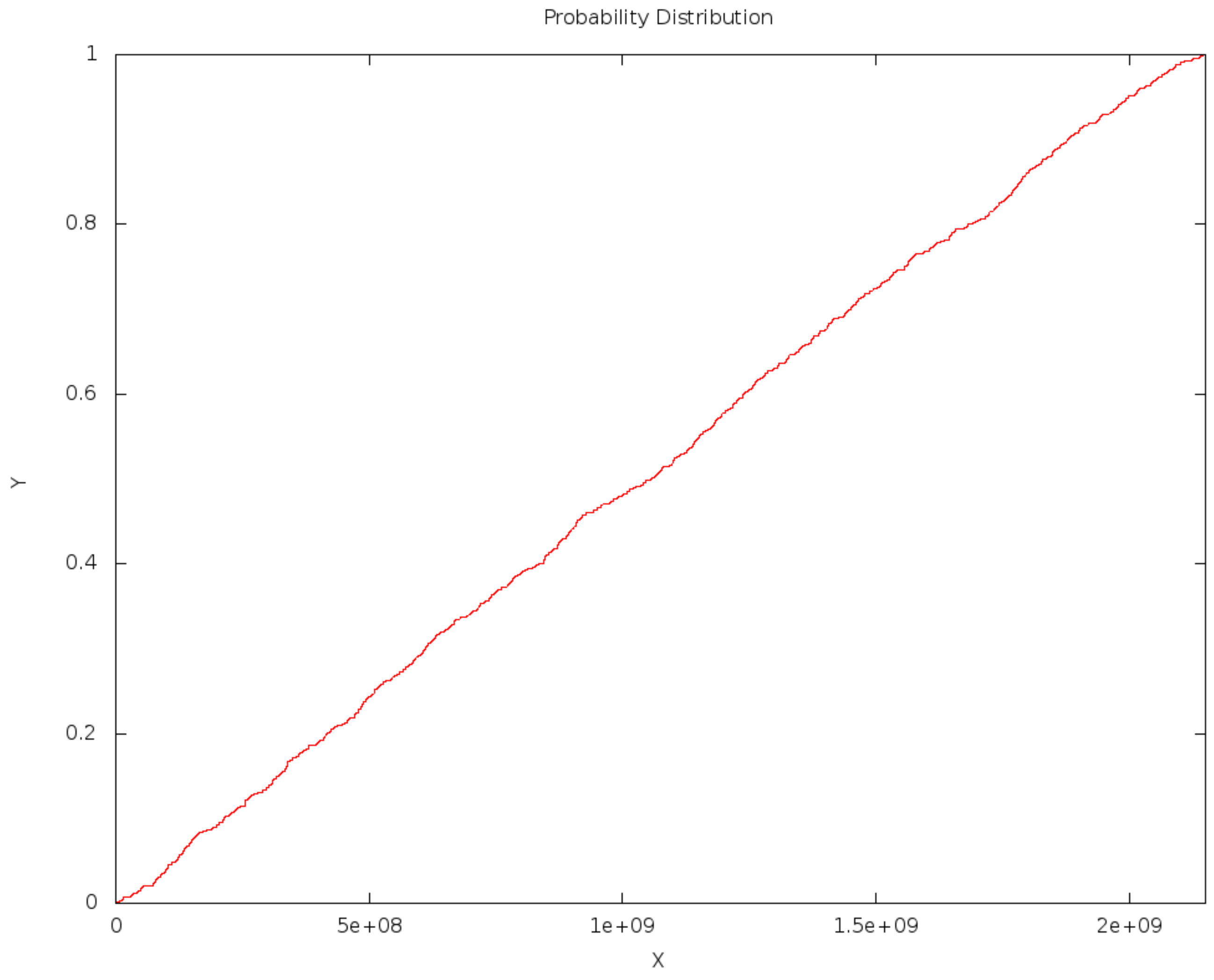


<u>Plot of (u_i, u_{i+1}) for a constant value of 'a' and 'm' with n=100000</u>

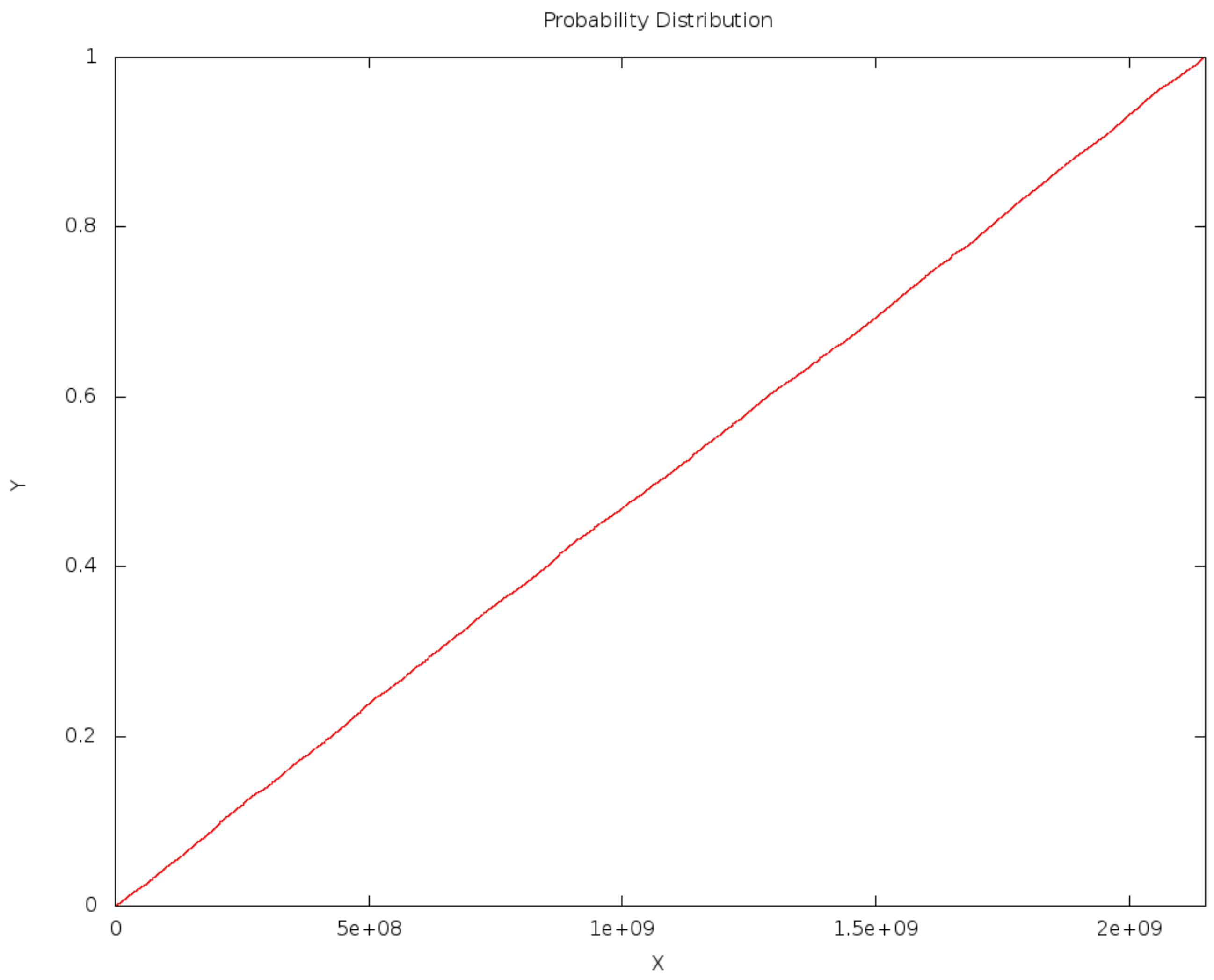
a = 4179 , b = 214 , m = 2147483563



Probability Distribution for n=1000



Probability Distribution for n=10000



OBSERVATIONS

Values of the Autocorrelation Lag for a=16807 , m= 2^31-1 and n=1000

The auto correlation of lag for 1 is 0.023901

The auto correlation of lag for 2 is -0.008424

The auto correlation of lag for 3 is -0.056108

The auto correlation of lag for 4 is -0.090982

The auto correlation of lag for 5 is -0.096801

Values of the mean and variance for a= 16807 , m= 2^31-1 and varying n

For n=1000:
Mean: 1043428075.521
Variance: 366502169583250496.00

For n=10000:
Mean: 1071586934.4838
Variance: 387471473674617088.00

For n=100000:
Mean: 1069887737.01859
Variance: 384644401825859008.00

RESULT

1. The autocorrelation lag decreases as the lag number increases i.e. autocorrelation lag is minimum for lag number equal to 5.

2. The sample mean lies in the range (1043428075.521, 1071586934.4838) ans will converge to some value in between for very large n. The sample variance lies in the range (366502169583250496.00, 384644401825859008.00). Similar as mean, variance would also converge to some value in between for very large n.

3. The extended Fibonacci generator passes most statistical tests which prove its efficacy.

4. The extended Fibonacci generator has a period of $2^k(2^{17}- 1)$ which is much longer than Linear Congruential Generators.

5. On generating the probability distribution,we find that the graph is a straight line with an positive slope. Also,the graph is monotonically increasing.

6. As the number of iterations increase, the slope of probability distribution graph becomes nearly straight.