Contents

General LCG and extended Fibonacci generator

Abhinav Gupta

February 8, 2017

1 Short Summary

"$y \ mod \ m$ can be implemented as $y - (y/m)\Delta m$".

It can be shown that to avoid overflow a straight forward implementation of a linear congruential generator in integer variables must be restricted to small modulus.

$$ax_i mod m = a(x_i mod q) - \lfloor x_i/q \rfloor r + (\lfloor x_i/q \rfloor - \lfloor ax_i/m \rfloor)m$$

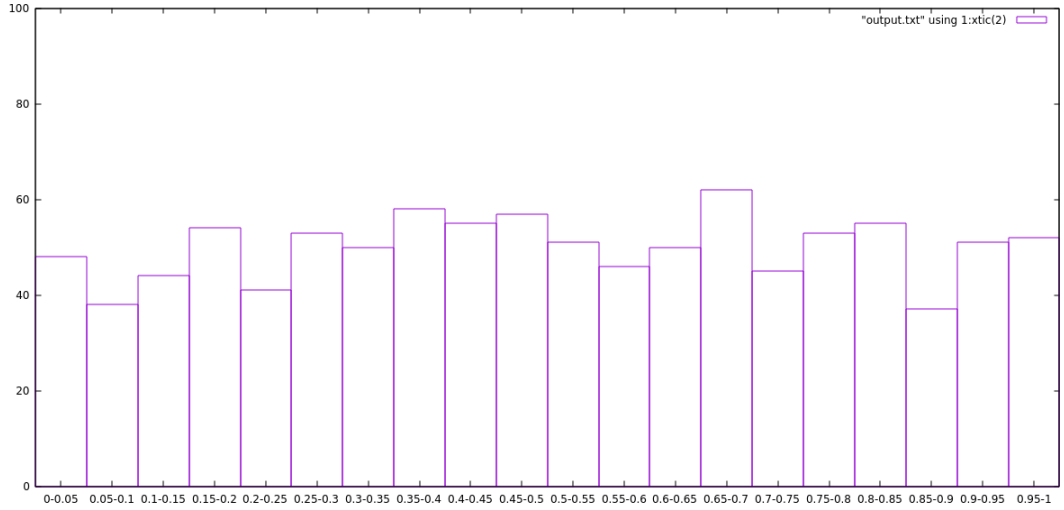
–> In this case the last term $(\lfloor x_i/q \rfloor - \lfloor ax_i/m \rfloor)m$ has value either 0 or m. Therefore the last term is m precisely when $a(x_i mod q) - \lfloor x_i/q \rfloor r < 0$.

2 Question-1

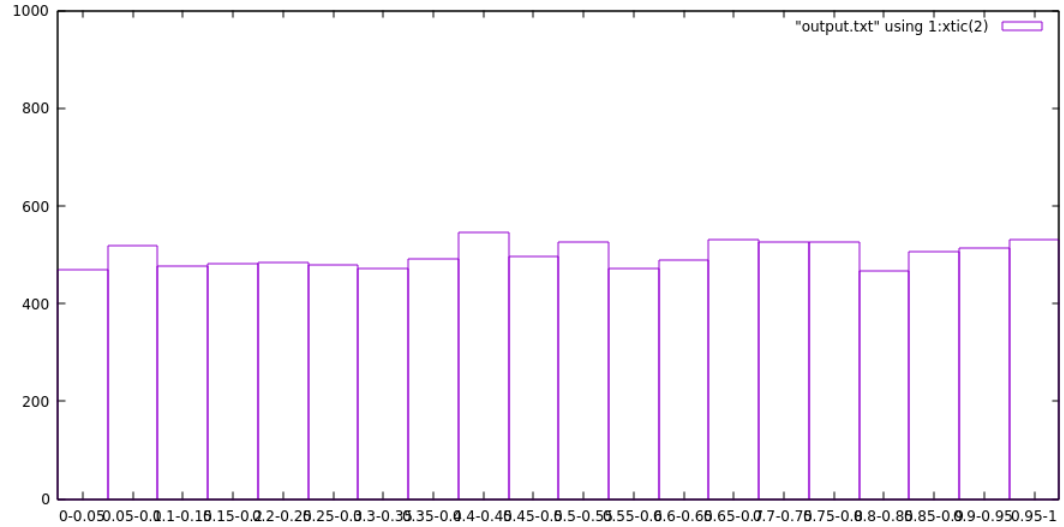
Uniform Look-alike distribution

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

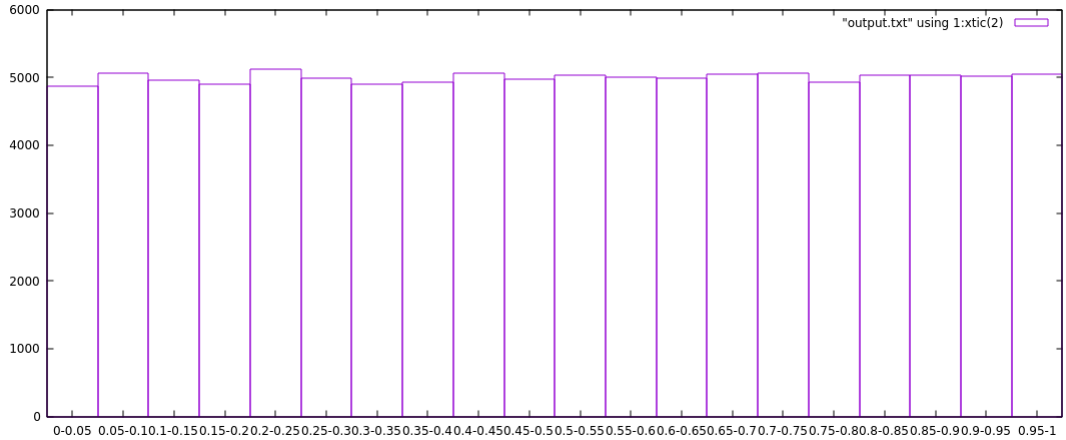
- a=16807 $m = 2^{31} - 1$ count= 1000



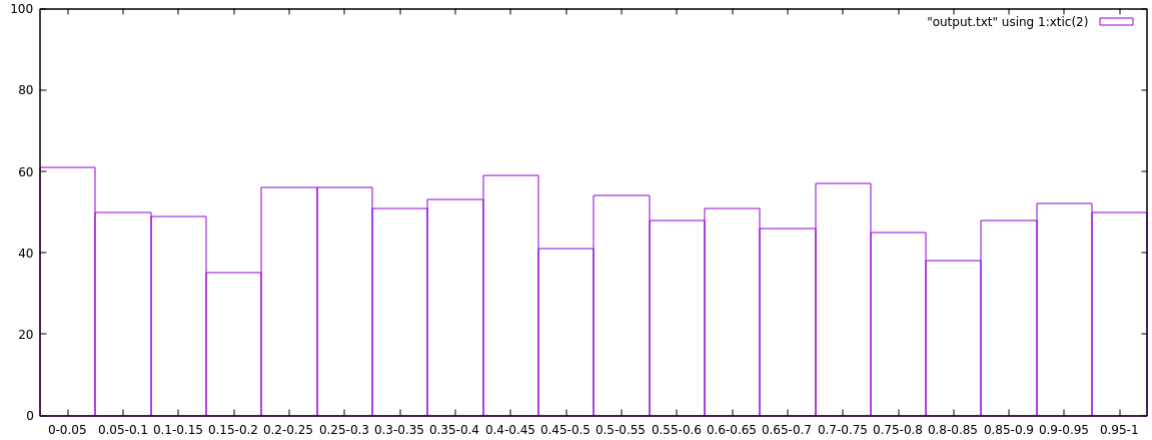
count= 10000



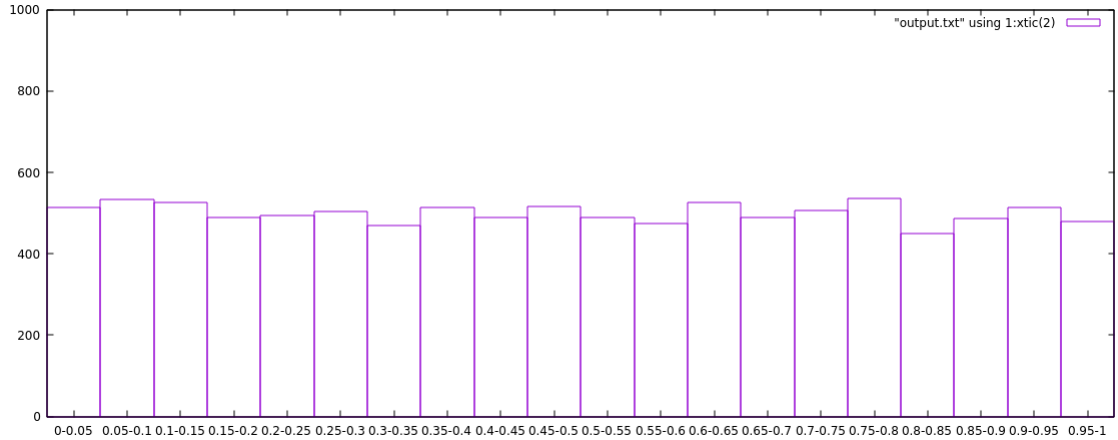
count= 100000



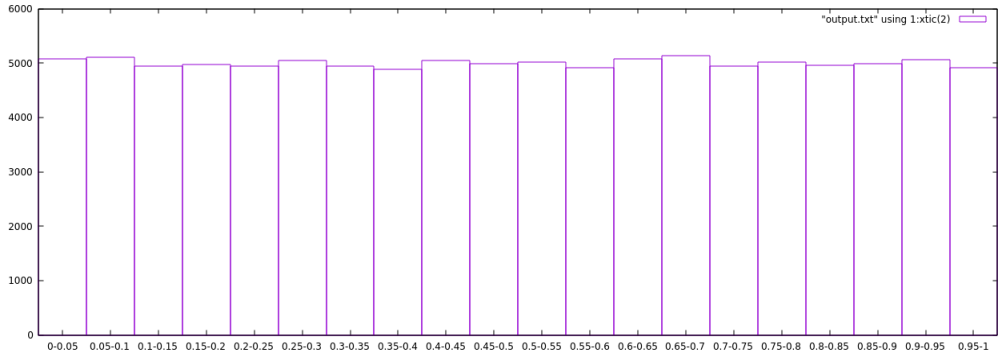
- a=40692 $m = 2147483399$ count= 1000



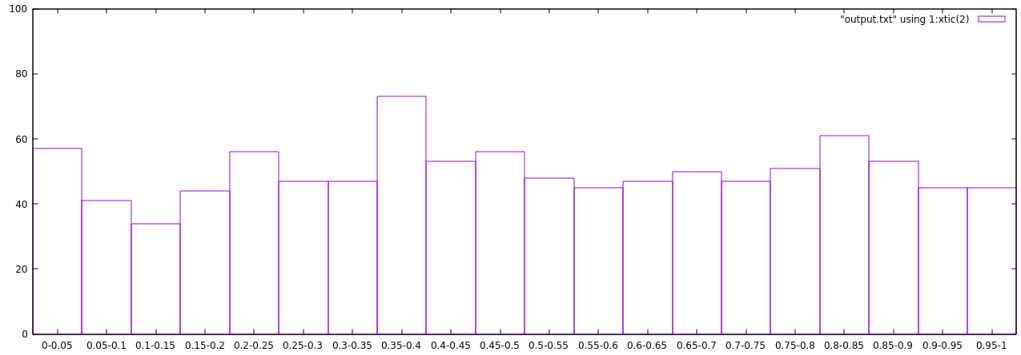
count= 10000



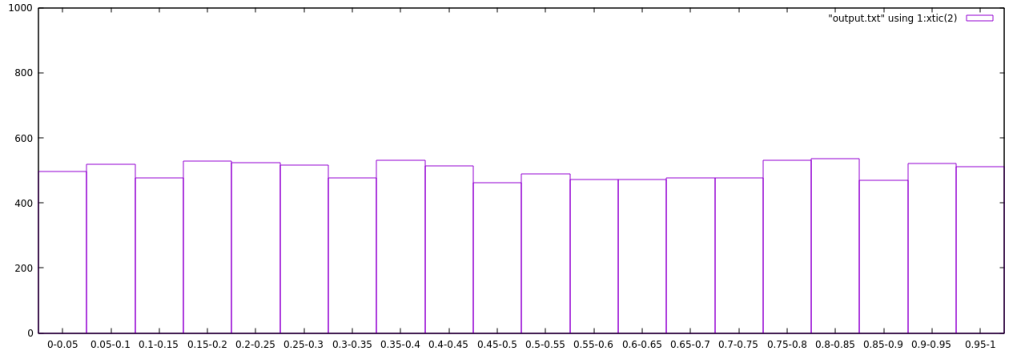
count= 100000



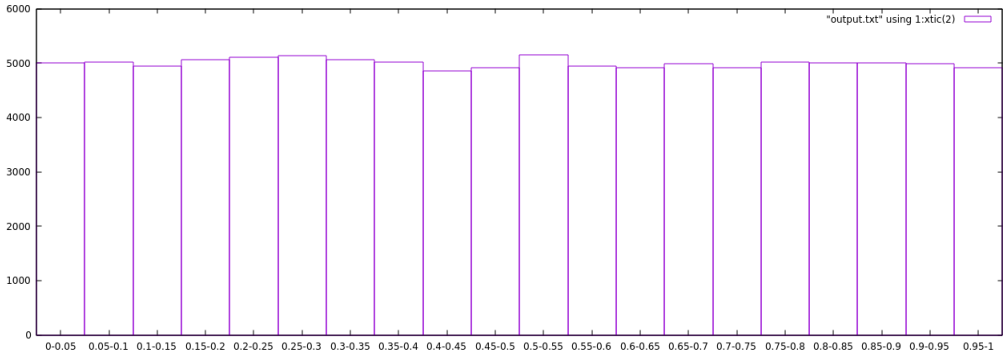
- a=40014 $m = 2147483563$ count= 1000



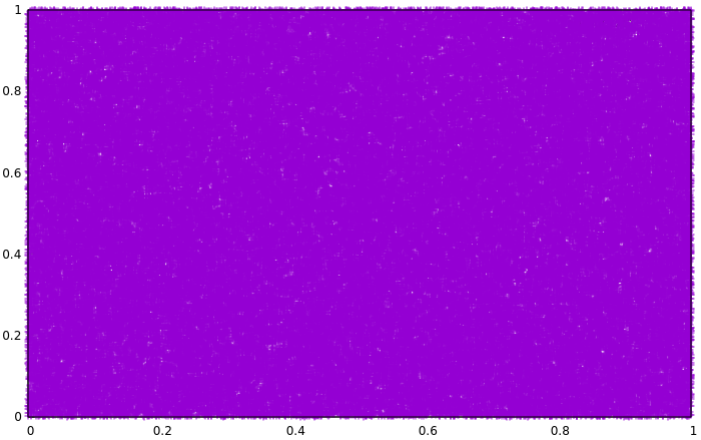
count= 10000



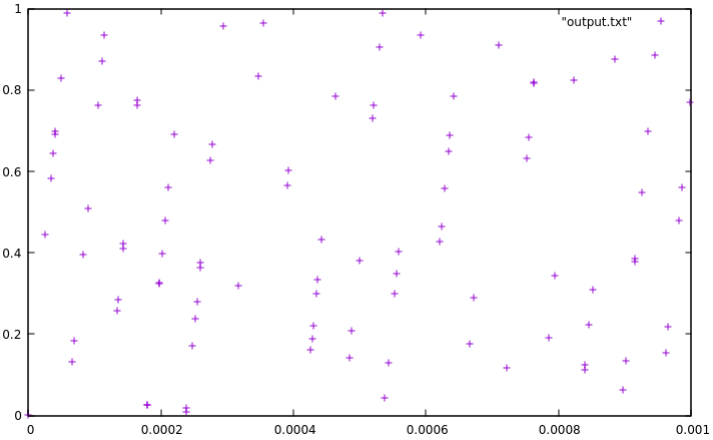
count= 100000



- **Plot of values** (u_i, u_{i+1}) **:** count=100000



- **Plot of values** u_i **from (0, 0.001) :** count=100000 After zooming it still looks like points are on a certain hyperplane but number of hyperplanes are quite good.



- **Code:**

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <algorithm>
int main(){
        long long int x;
        int a,cnt[20]={0},i,k;
        double m;
        x=23;
        a=16807;
        m=pow(2,31)-1;
        int count=100000;
        float u[count],j;
        for(i=0;i<count;i++){
                u[i]=x/m;
                x=(x*a);
                x%=(long long int)m;

                for(k=0,j=0.05;k<20;k++){
                        if(u[i]<j){
                                cnt[k]++;
                                break;
                        }
                        j+=.05;
                }
        }
        for(i=0;i<(count-1);i++){
                printf("%f %f\n",u[i],u[i+1]);
        }
        printf("\n\n\n");
        for(k=0,j=0.00;k<20;k++,j+=0.05){
                printf("%d %g-%g\n",cnt[k],j,j+0.05);
        }
}
```

3 Question-2

$$R_k = (\sum_{t=k+1}^{T} (Y_t - \bar{Y})(Y_{l-k} - \bar{Y}))/(\sum_{t=1}^{T}(y_t - \bar{Y})^2)$$

Since the denominator and numerator contain different number of terms, in particular as the number of terms in the numerator is decreasing in time lag "K". Thus the estimator looks biased and for the large values of k R_k will vanish.**But this is what actually wanted.**

Plus autocorrelation of Fibonacci generator is more close to zero than that of LCG. Therefore Fibonacci generator is better than the given LCG in terms of autocorrelation.

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.
The sample mean for 1000 values is 0.493665.
The sample variance for 1000 values is 0.0827854.
Code:

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <algorithm>
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)\n", u[i-1], u[i]);
  }
```

```
}

float mean(float * u, int n){
  float m = 0;
  for (int i = 0; i < n; i++) {
    m += (u[i]/(float)n);
  }
  return m;
}

float variance(float * u, int n, float mu){
  float v = 0;
  for (int i = 0; i < n; i++) {
    v += (pow((u[i]- mu),2)/(float)n);
  }
  return v;
}

// For Auto-Correlation
float rho(float * u, int n, int l, float mu){
  float num = 0, denom = 0;

  for (int t = l+1; t <= n; t++) {
    num += ((u[t] - mu) * (u[t - l] - mu));
  }

  for (int t = 1; t <= n; t++) {
    denom += pow((u[t]- mu),2);
  }

  return (num / denom);
}

int main(){
  float u[1000000];
  // Random Numbers generated by list constructor
  list(100000, u);

// certain number of values printed from same set of generated random numbers
  printf("Plot data :");
  printf("\n\t#For 1000 values ::\n");
  data(u, 1000);

  printf("\n\t#For 10000 values ::\n");
  data(u, 10000);
```

```
  printf("\n\t#For 100000 values ::\n");
  data(u, 100000);
  printf("\n");

  float mu = mean(u, 1000);
  printf("\nThe sample mean for 1000 values is %g.\n", mu);

  float var = variance(u, 1000, mu);
  printf("\nThe sample variance for 1000 values is %g.\n\n", var);

//Autocorrelation of lags
  for (int i = 1; i <= 5; i++) {
    printf("Autocorrelation of lag %d, with 1000 generated values, is %g.\n", i, rho(u, 1000
  }
}
```