Lecture 3 🎋

Class: Random number Generation

Date: Today , 13:17

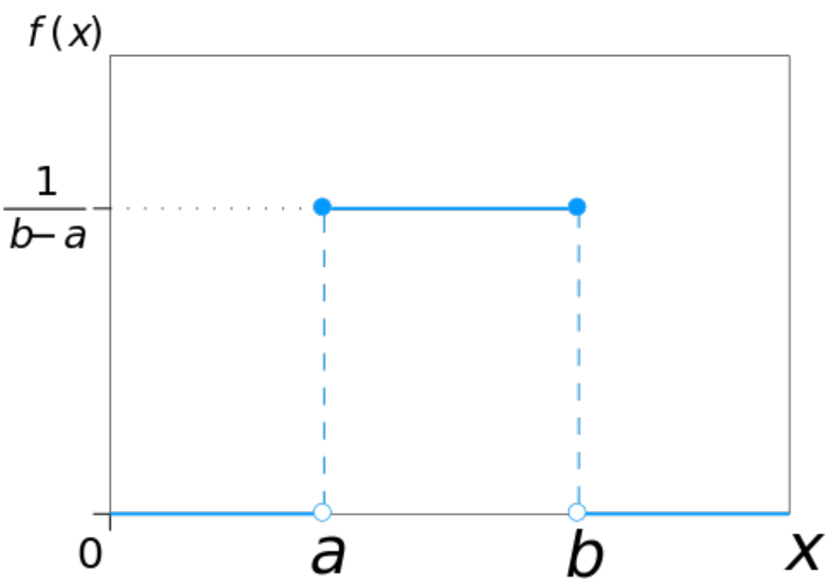
Author: Lasal Hettiarachchi

Key learnings:

- Linear Congruential method
- Multiple recursive generator

Uniform random number generation

- The probability of getting all the numbers between a and b is same



- Computers cannot generate random numbers
- So we give a seed and follow an algorithm

- such algorithms can be represented as a tuple (S, f, μ, U, g), where
  - S is a finite set of **states,**
  - f is a function from S to S,
  - μ is a probability distribution on S,
  - U is the **output space;** for a uniform random number generator U is the interval $(0,1)$, *and we will assume so from now on, unless otherwise specified,*
  - g is a function from S to U.

6

Linear Congruential method

x0 = 1

m = 5 (period)

a= 10

c = 3

x1= (10(1)+3)mod 5 = 3

x2 = (10(3) +3) mod 5 = 3

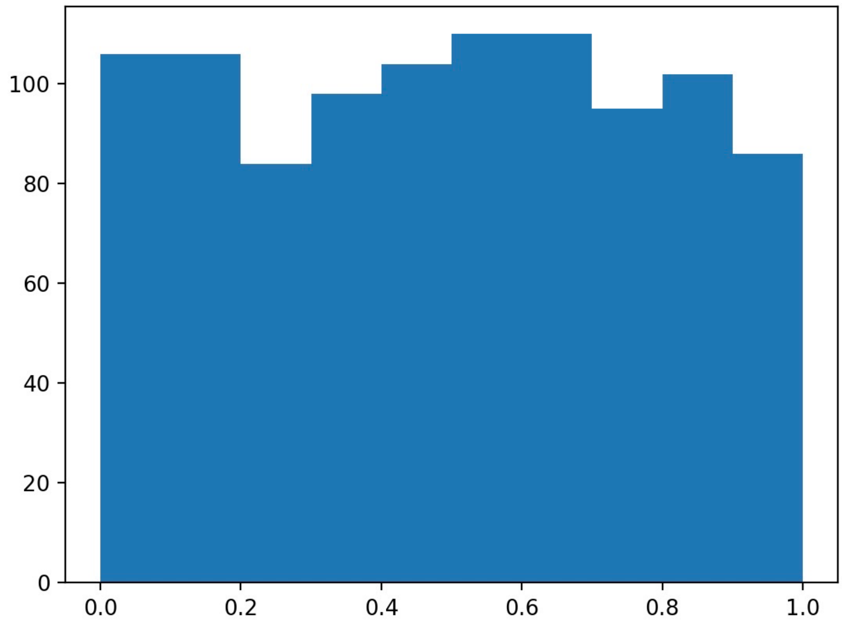
we are getting 3s here , since m,a and c are very small but in actual scenrarios they are very large.

x0 can only change between 0 and m-1

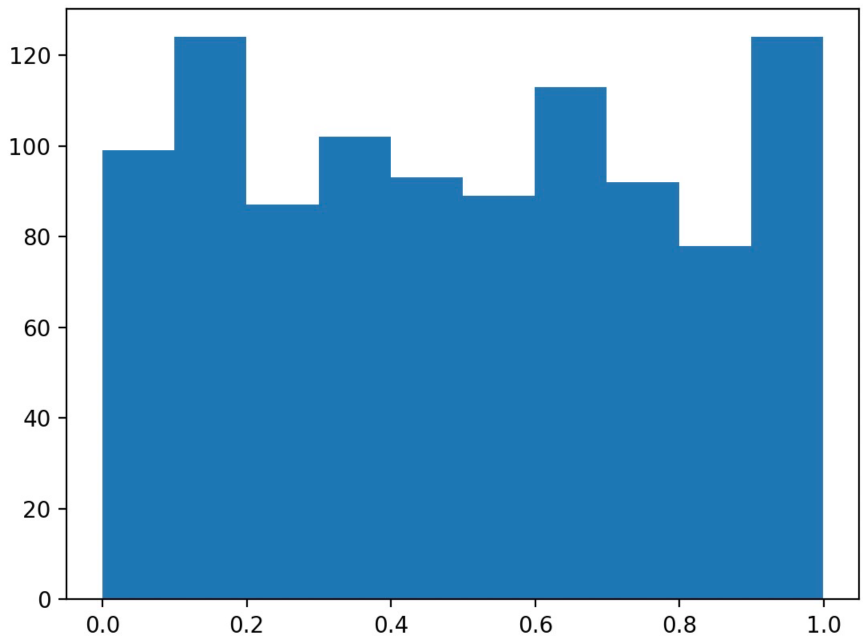
```
#implement linear congruential generators
# x_{n+1} = (a*x_n + c) mod m
x_0 = 1000
a = 7 ** 5
c = 0
m = 2**31 - 1
Xtarray = [x_0]
UtArray = [x_0/m]
for i in range(0,1000):
    x_0 = (a*Xtarray[i] + c) % m
    Xtarray.append(x_0)
    UtArray.append(x_0/m)

print("xt array:",Xtarray)
print("Ut array:",UtArray)

#plot Ut array
import matplotlib.pyplot as plt
# plt.plot(UtArray)
# plt.show()

#plot histogram
plt.hist(UtArray)
plt.show()
```

---

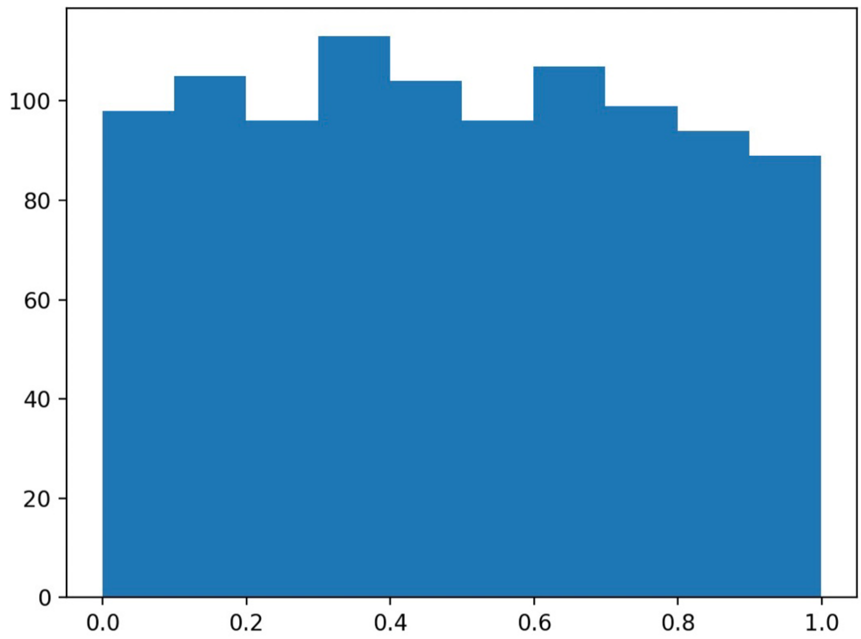
x0 = 1000


0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0

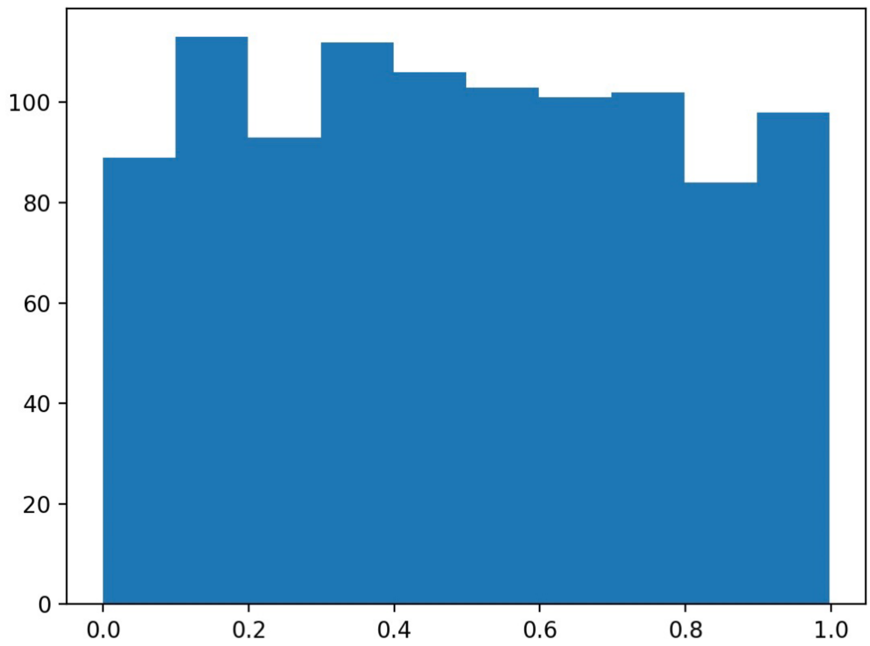
x0 = 100


0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0

x0 = 5



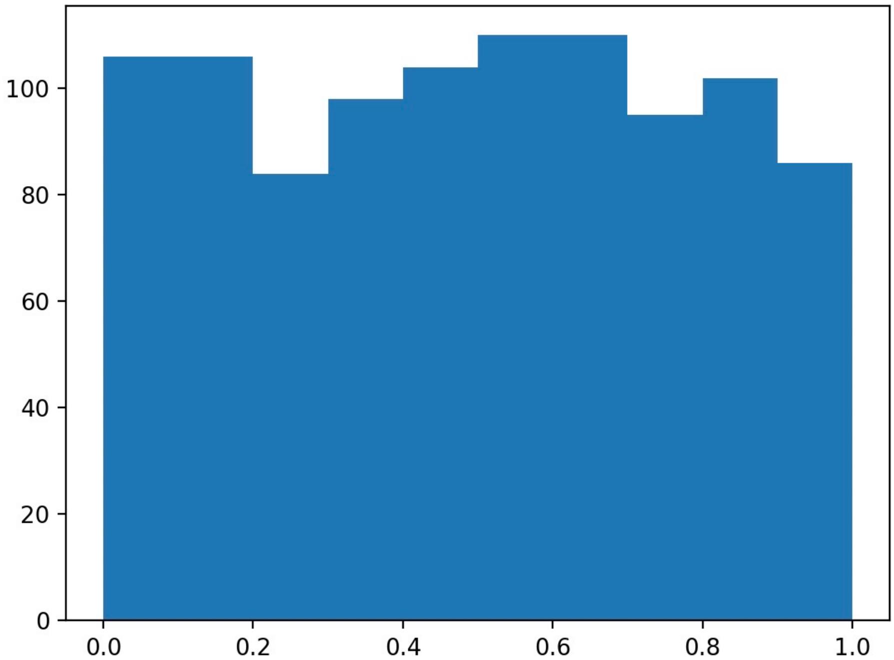
x0 = 10000



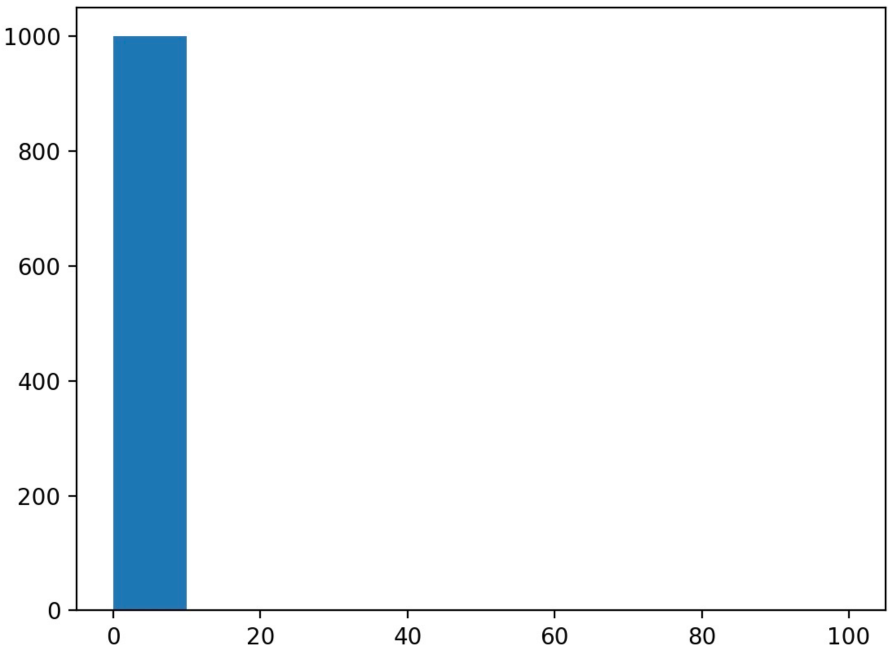
The frequency distribution does not change with the value for x0. Rather the uniform distribution holds despite the values of x0 chaning. We can further observe that the value for the random numbers fluctuate between 0 and 1

---

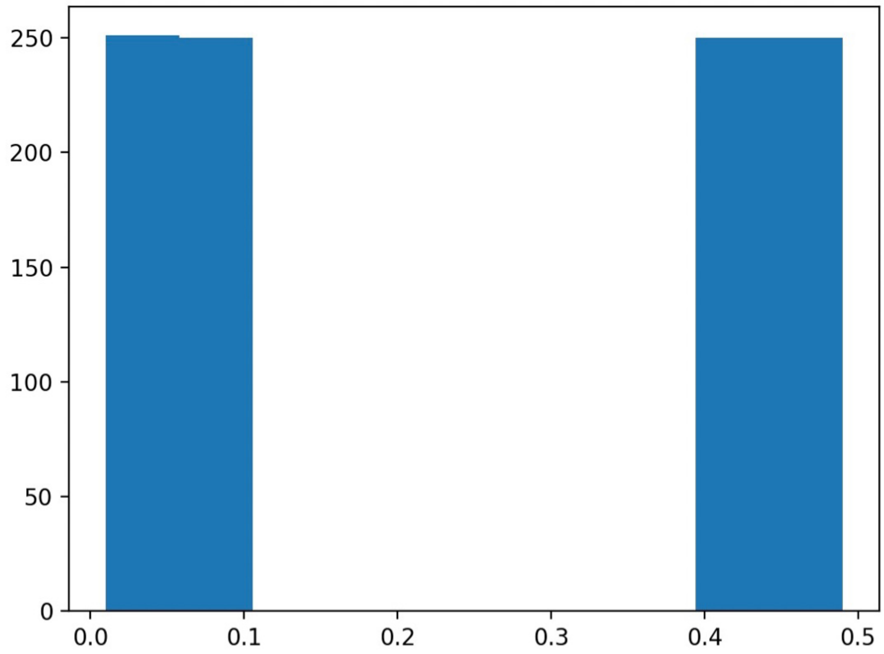
m0 = 2 ** 31 -1


100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

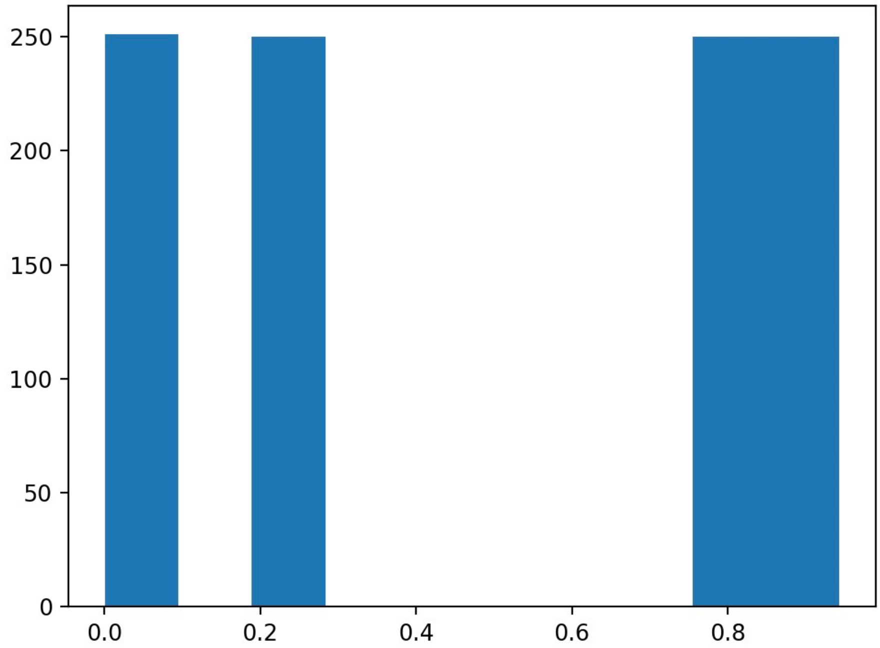
m0 = 10


1000
800
600
400
200
0
0
20
40
60
80
100

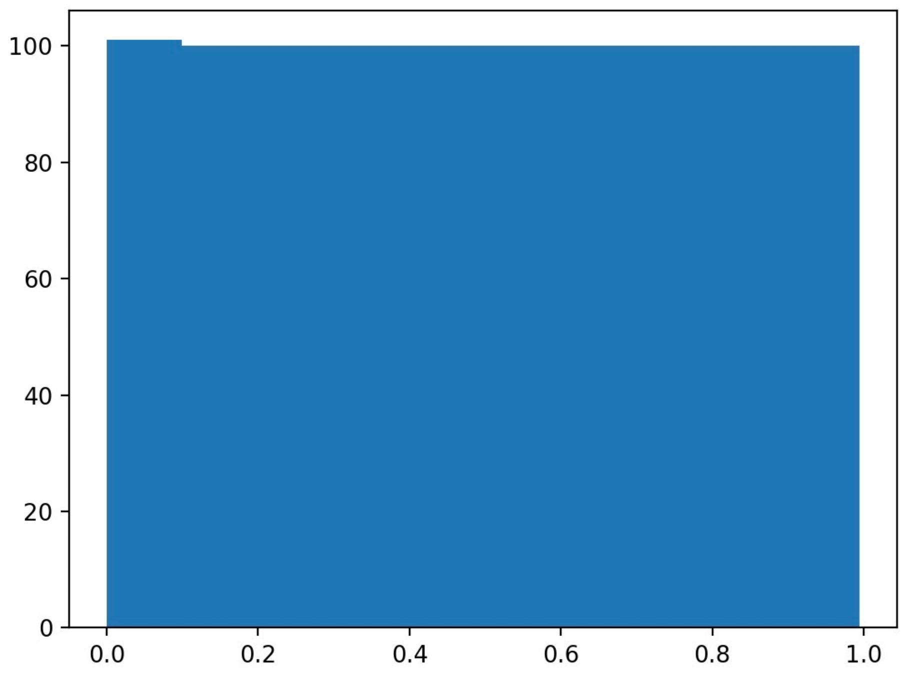
m0 = 10 ** 5


250
200
150
100
50
0
0.0
0.1
0.2
0.3
0.4
0.5

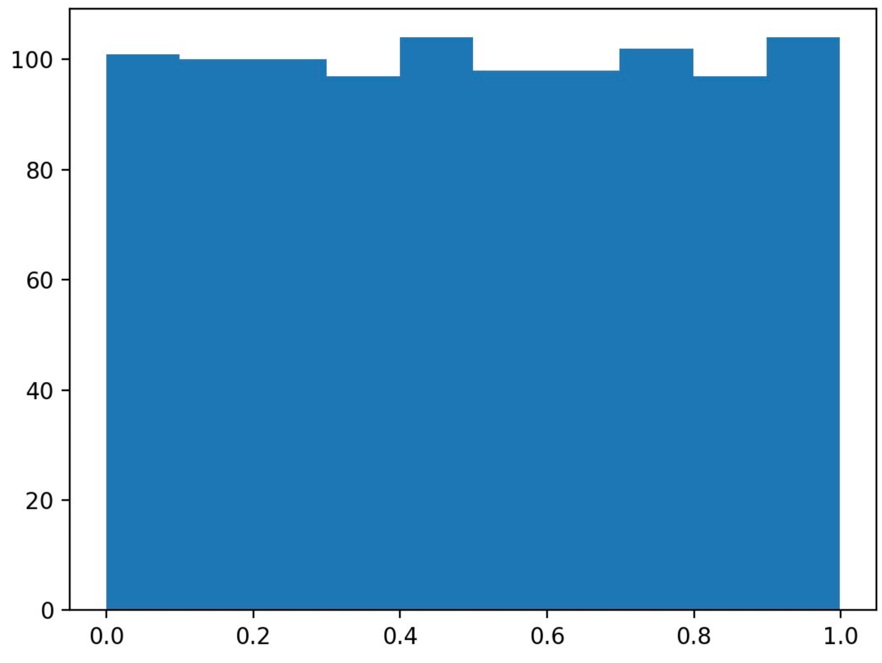
m0 = 10 ** 6


250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8

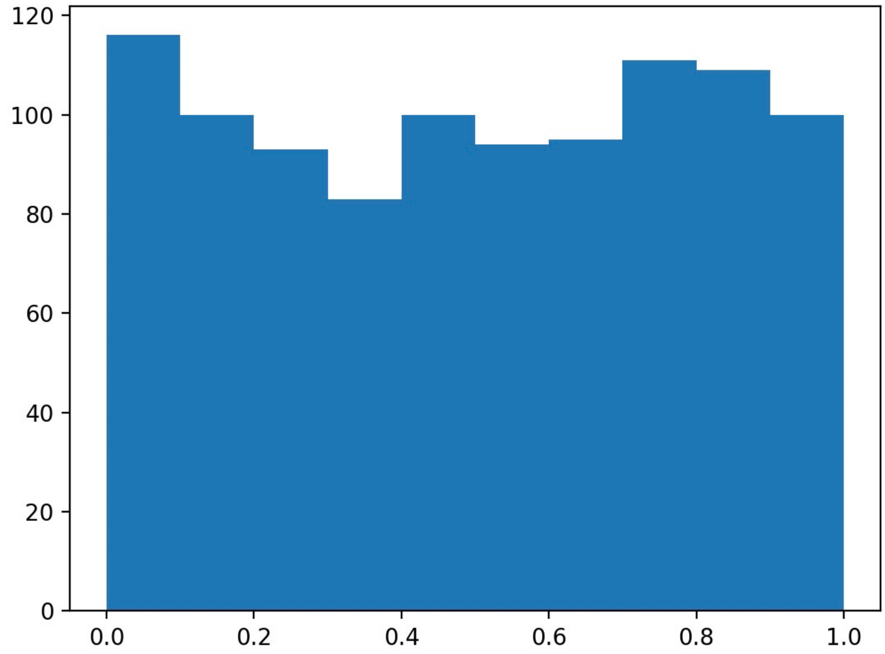
m0 = 10 ** 7



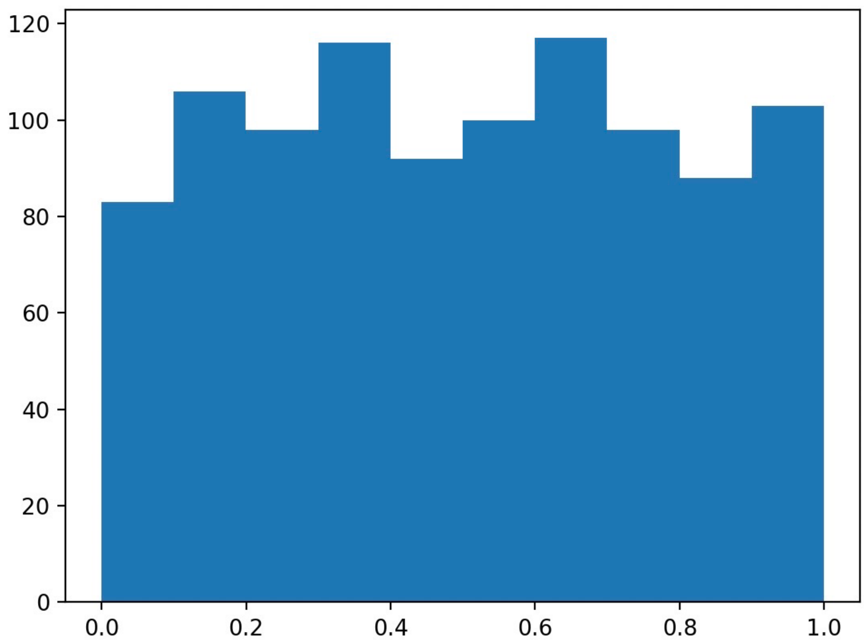
m0 = 10 ** 10



m0 = 10 ** 31



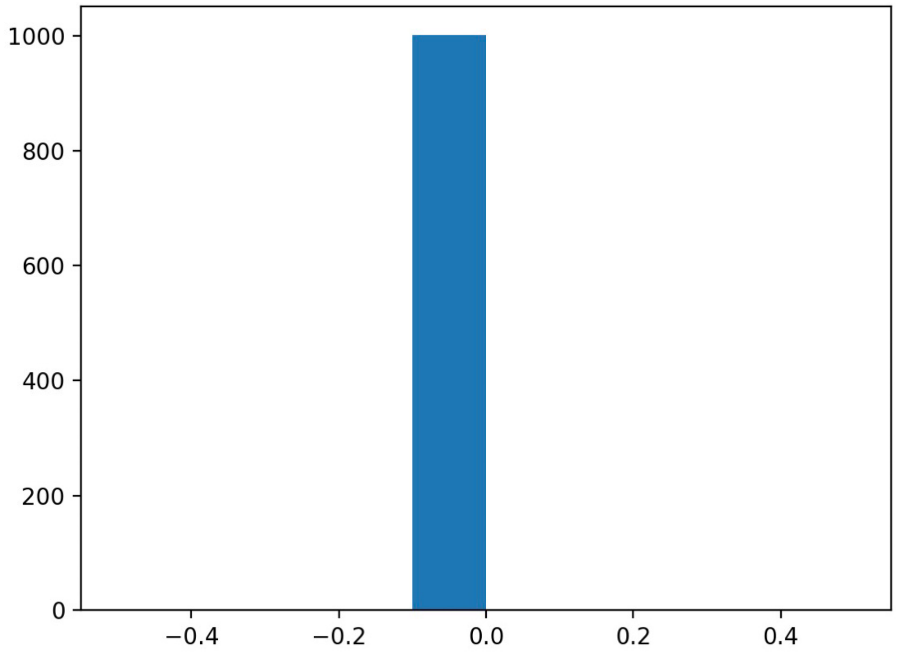
m0 = 10 ** 40



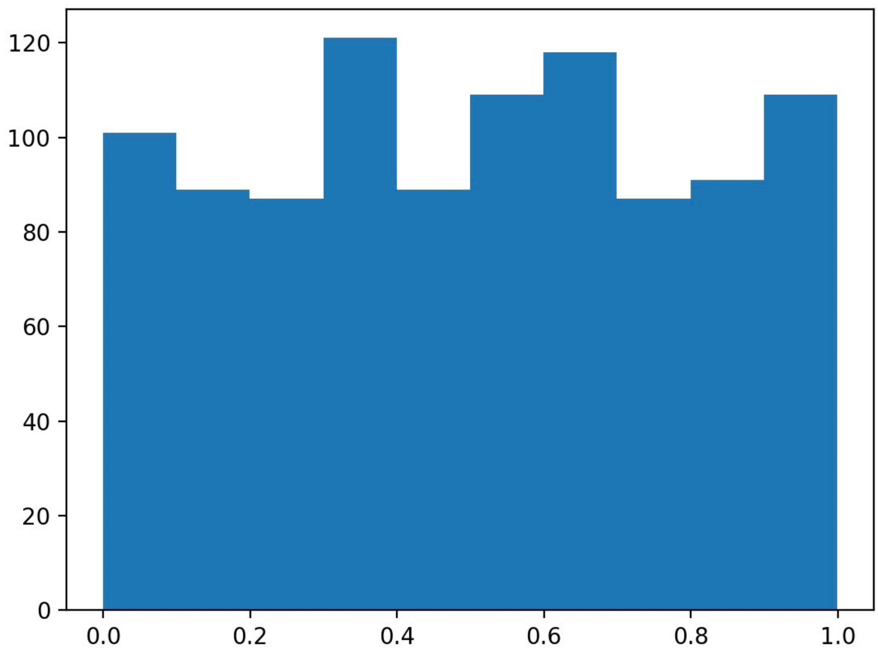
The frequency distribution depends on the m0 value. The uniform frequency distribution is not obtained until the m0 value passes a specific value. Just after the it passes that value. the uniformity of the distribution is extremely high (see 10**7 where the 1000 values are distributed with each range having approximately 100 entries). By increasing the m0 value further, a uniform frequency distribution can be obtained

---

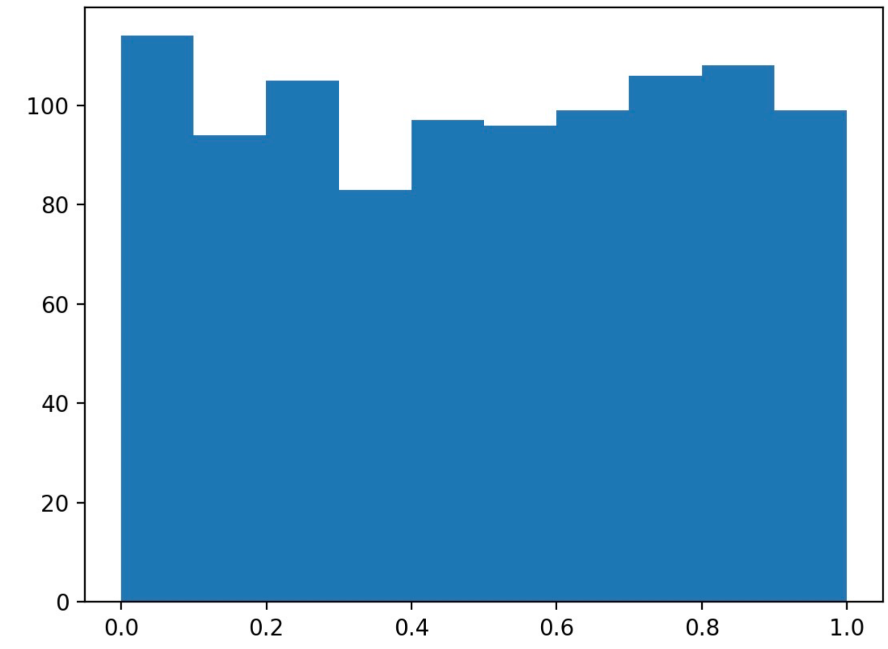
a = 1


1000
800
600
400
200
0
−0.4
−0.2
0.0
0.2
0.4

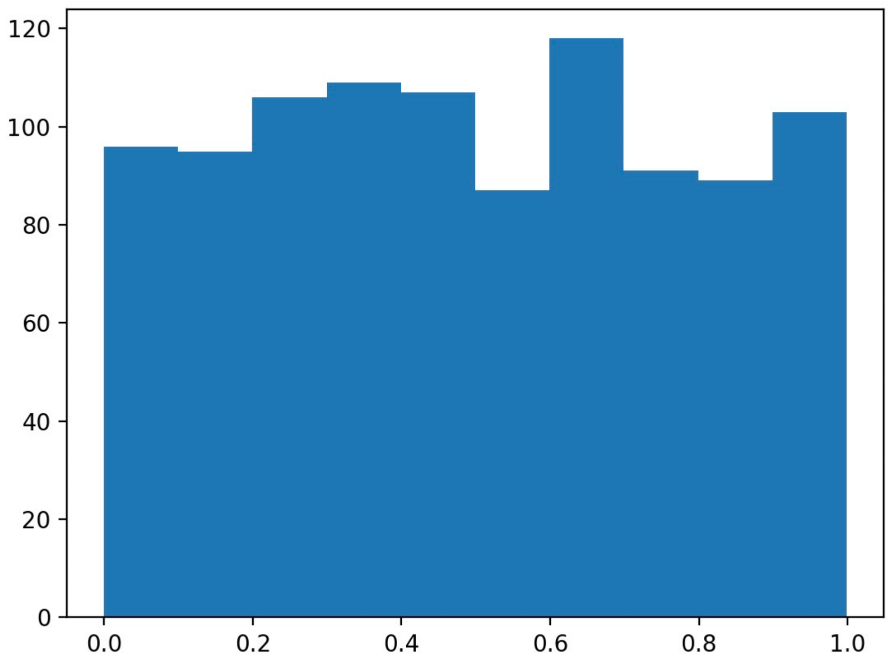
a = 10


120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

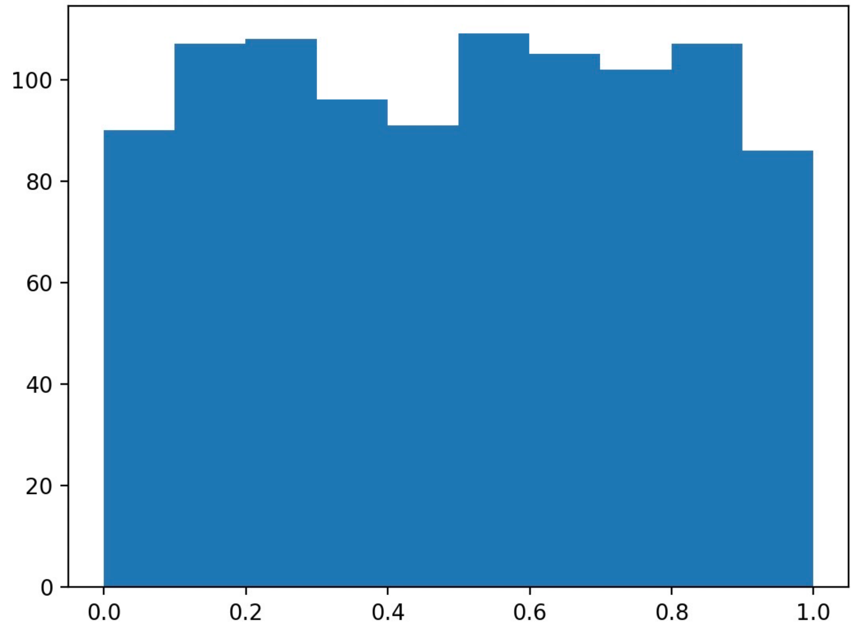
a = 7 ** 3


100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

a = 7 ** 10


120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

a = 7 ** 1000



The uniformity of the frequency distribution holds for all a values other than 1

Multiple recursive generator

In here the first 1000 random values were calculated using Linear Congruential method

- A **multiple-recursive generator** (MRG) **of order** k; is a random number generator of the form of Algorithm 1, with state $S_t = \mathrm{X}_t = (X_{t-k+1}, \dots, X_t)^T \in \{0, 1, 2, \dots, m-1\}^k$ for some modulus m and state transitions defined by

$$\mathrm{X}_t = (a_1 X_{t-1} + \cdots + a_k X_{t-k}) mod\ m, \qquad t = k, k+1, \dots$$

where the **multipliers** $\{a_i, i = 1, \dots, k\}$ lie in the set $\{0, 1, 2, \dots, m-1\}$

- The output function is often taken as

$$U_t = \frac{X_t}{m}$$

```
#implement multiple recursive generators
# x_{n+1} = (a_1*x_n + a_2*x_{n-1} + ... + a_k*x_{n-k+1} + c) mod m
# Using Linear method to generate the first 10 values of the sequence
x_0 = 1000
a = 7 ** 5
c = 0
m = 2**31 - 1
Xtarray = [x_0]
UtArray = [x_0/m]
for i in range(0,999):
    x_0 = (a*x_0 + c) % m
    Xtarray.append(x_0)
    UtArray.append(x_0/m)

# Using the first 10 values to generate the next 1000 values
# define a array with 1000 elements as 7 **5
a = [7 ** 5] * 1000
m = 2**31 - 1

for t in range(1001,2001):
    sum = 0
    for i in range(0,1000):
        sum = sum + a[i] * Xtarray[t-i-2]
    x_0 = (sum) % m
    Xtarray.append(x_0)
    UtArray.append(x_0/m)

print("xt array:",Xtarray)
print("Ut array:",UtArray)


#plot histogram
import matplotlib.pyplot as plt
plt.hist(UtArray)
plt.show()
```