Kolmogorv-Smirnov Test

Computer-Intensive in Statistics Project I Report

Scott Williams

1 Introduction

1.1 Goal of the Project

We want to perform the Kolmogorov-Smirnov test on a set of random numbers generated and see if our sample passes the test. Furthermore, we want to attempt to answer whether the random number generator is truly generating the uniform distribution on the unit interval.

2 Approach

2.1 Programming Environment & Preparation

To perform the Kolmogorov-Smirnov test, we will be using a Python 3 environment. The necessary dependencies for the code are the following Python libraries: math, matplotlib, and numpy. To perform the test, we will first need to construct a function for random number generation and a function to construct the empirical distribution function. After those have been made, we can create an additional function that will perform the Kolmogorov-Smirnov test. It is important that we approach this problem in a modular way. Doing so will allow us to perform many iterations of the test and see if we are generating from the standard uniform distribution with various different seeds.

2.2 Random Number Generator

We will be using the function titled pseudo_random_uniform to perform pseudo random number generation. The function takes the initial seed value chosen and the sample size as arguments. First an empty list titled values is defined. This will be how we store our random numbers. Next, we generate our random values using the following process.

1. Pick a seed
2. Evaluate: $X_{next} = 7^5$*seed mod(2^{31} - 1)

3. Evaluate: random value $= \frac{X_{next}}{2^{31}-1}$
4. Append random value to values list
5. Define the seed value to be X_{next}
6. Repeat 2, 3, 4, and 5 until the desired amount of numbers have been generated

To perform this process in Python, we calculate these values within a for loop. The values list is returned from the function so that we can use it for the Kolmogorov-Smirnov test.

2.3 Empirical Distribution Function

We will be using a function titled sample_distribution to construct the EDF. The function takes the values list generated by pseudo_random_uniform and the sample size as arguments. First, we take the list of random numbers and sort it from smallest to largest. Then we create an empty list S that will store the values for the EDF. We determine the EDF values based on the following criteria.

$$S(x) = \begin{cases} 0 & x < x_1 \\ \frac{r}{n} & x_r \leq x < x_{r+1} \\ 1 & x_n \leq x \end{cases}$$

Clarifying notation, r is the index, n is the sample size, x_1 is the smallest number, and x_n is the largest number. Once we have determined the appropriate value, we append it to S. In Python, we perform this process in a for loop for each random number in the values list. The function returns the list S so that it can be used for the Kolmogorov-Smirnov test.

2.4 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test tells us that if the random values generated are truly coming from the uniform distribution, then, with probability 0.99 the following equation will be true for n larger than 80.

$$sup_x|S_n(x) - F(x)| < \frac{1.6276}{\sqrt{n}}$$

Clarifying notation, $S_n(x)$ is the EDF, F(x) is the theoretical CDF that we are comparing to, and n is the sample size. Since we are comparing to the standard uniform distribution, F(x) = x. In our case, the values for F(x) will be equal to the values generated by our random number generator. From now on, the left side of the above inequality will be referred to as the test parameter and the right side of the above inequality will be referred to as the test value. We find the results of this test by constructing a function called kolmogorov_smirnov. The function takes the values list, the S list, and the sample size as arguments. We first calculate the test value and create an empty list called diff. Using a for loop, for each value of the EDF, we compute the absolute value of the difference with the CDF and append it to diff. After that process is complete, we find the maximum value

of diff and use it as our test parameter. Finally, we get the result of the Kolmogorov-Smirnov test by checking if our test parameter is less than our test value. If so, our sample has passed the test.

3 Performing the Kolmogorov-Smirnov Test

3.1 Testing on One Sample

Since our results will be from running the code for the test in Python, it should be noted that all code and output will be included at the end in section 5 titled Figures and Python Code. For this initial test, we will be using 1 for the seed and 1000 for the sample size. Using each function described in the previous section, we find that for this specific case, the sample does pass the Kolmogorov-Smirnov test. To visualize these results, we make a histogram of the sample and a plot with both the EDF and CDF. A histogram of the standard uniform distribution should approximately be the shape of a rectangle. Observe that we obtain these results in the histogram of the random numbers we have generated. Furthermore, on the EDF and CDF plot, we can see that the functions produce similar curves as there is not much fluctuation between the CDF and EDF. To further inspect our sample, we use Monte Carlo techniques to estimate the mean, variance, and standard deviation. For simplicity of notation, we will refer to the values in our sample as x_i. To calculate the mean, we approximate the expected value by calculating the long run average.

$$\frac{\sum_{i=1}^{1000} x_i}{1000} \approx \ 0.4979$$

To approximate the variance, we perform the same Monte Carlo calculation on x_i^2 and subtract the mean^2.

$$\frac{\sum_{i=1}^{1000} x_i^2}{1000} - mean^2 \approx 0.078$$

Finally, to approximate the standard deviation, we take the square root of the variance.

$$\sqrt{variance} \approx 0.2806$$

For the standard uniform distribution, the mean = 0.5, the variance = 0.08, and the standard deviation = 0.28. The approximations we calculated using Monte Carlo techniques for our sample are very close to the true values. Our sample passes the Kolmogorov-Smirnov test and has the expected characteristics of the standard uniform distribution.

3.2 Testing on Multiple Samples

We now want to answer if the generator we are using is truly generating numbers from the standard uniform distribution. Although the sample in the last section passed the Kolmogorov-Smirnov test

and had promising characteristics, that was just one sample. We would like to further study the generator by using different seeds and find the percentage of samples that pass the Kolmogorov-Smirnov test. To do this, we first create an altered function for the Kolmogorov-Smirnov test that will suppress output unless the test fails. The function returns 1 if the sample passes the test and 0 if it fails. In Python, this function is named ks_suppressed and takes the same arguments as the kolmogorov_smirnov function that was described earlier. We will then use ks_suppressed in a function called generator_test that takes the amount of trials we wish to run as an argument. The different seeds will be stored in a list of the same size as the number of trials. The seeds used will be evenly spaced numbers from 0 to number of trials minus 1 since Python indexes from 0. In a for loop, we perform the Kolmogorov-Smirnov test for each different seed and keep a running total of how many times we have passed the test. To try to gain more insight, if our sample fails the test, we print out a notice that the test failed, the test parameter, the test value, the seed that produced failure, the lower and upper bound of the failing sample, the estimated mean, the estimated variance, the estimated standard deviation, the histogram of the sample, and the plot with the EDF and CDF. After everything has been performed, we print out the percentage of passes.

We perform the generator test with 1000 different seeds and observe that it fails for 7 different seeds. From the results, the failure that stands out the most is when our seed is 0. Looking back at our random number generator, it is clear that 0 is not a good choice of seed. This is because we are continuously calculating 0 mod (2^{31} - 1) which is always 0. Therefore, we have a sample of 1000 zeros which is hardly random. All of the other failures have characteristics such as lower bound, upper bound, mean, variance, and standard deviation that are close to what we would expect from the standard uniform distribution. However, looking at the plots of the EDF and CDF we can see that there is clearly a difference between the two curves. With 1000 different seeds, our generator passes the Kolmogorov-Smirnov test in 99.3% of the trials.

4 Conclusion

4.1 Final Thoughts

Recall, that the Kolmogorov-Smirnov test states that for n $>$ 80, if the random values generated are truly coming from the standard uniform distribution, then, the inequality will be true with probability 0.99. The results of our generator test are consistent with this because we found for n = 1000 the inequality in the Kolmogorov-Smirnov test was true for 99.3% of the trials. This means that the random values we generated were truly coming from the standard uniform distribution. Therefore, the generator does generate the uniform distribution on the unit interval.

5 Figures and Python Code

Importing dependencies

In [1]:
```
# Importing the necessary libraries
import math
import matplotlib.pyplot as plt
import numpy as np
```

Function definitions

In [2]:
```
# Defining a function that will generate pseudo random numbers with the specified generator
def pseudo_random_uniform(seed, size):
    values = []
    for i in range(size):
        x_next = (7**5)*seed % ((2**31) - 1)
        random_value = x_next/((2**31) - 1)
        values.append(random_value)
        seed = x_next
    return values
```

In [3]:
```
# Defining a function that will give the value of the sample distribution function at each random number generated
def sample_distribution(values, size):
    ordered = values
    ordered.sort()
    S = []
    for i in range(size):
        if(i < (size - 1)):
            if(values[i] < ordered[0]):
                S.append(0)
            elif(ordered[i] <= values[i] and values[i] < ordered[i+1]):
                S.append(i/size)
            elif(ordered[(size - 1)] <= values[i]):
                S.append(1)
        if(i == (size - 1)):
            if(values[i] < ordered[0]):
                S.append(0)
            elif(ordered[i] <= values[i]):
                S.append(i/size)
            elif(ordered[(size - 1)] <= values[i]):
                S.append(1)
    return S
```

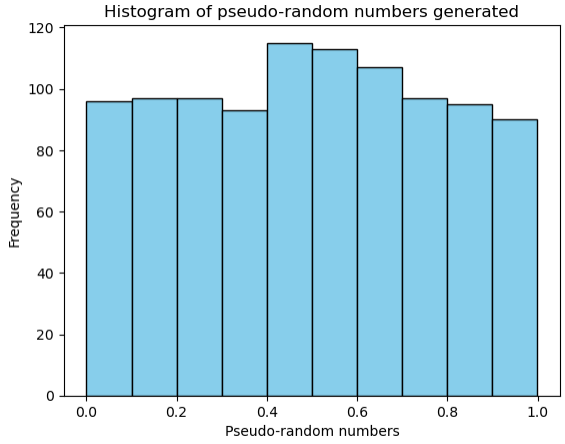
In [4]:
```
# Defining a function that will perform the Kolmogorov and Smirnov test
def kolmogorov_smirnov(values, S, size):
    test_value = 1.6276/math.sqrt(size)
    diff = []
    for i in range(size):
        diff.append(abs(S[i] - values[i]))
    test_param = max(diff)

    print("The test parameter is: " + str(test_param))
    print("The test value is: " + str(test_value))

    if(test_param < test_value):
        print("This distribution passes the Kolmogorov and Smirnov Test")
    else:
        print("The test failed")
```

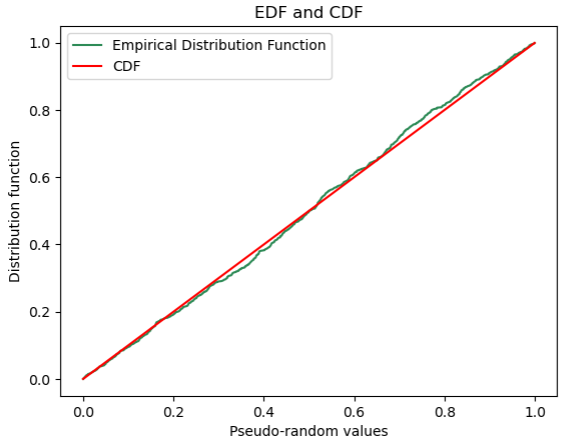
Testing on One Sample

In [5]:
```
# Running the test
values = pseudo_random_uniform(1, 1000)
S = sample_distribution(values, 1000)
kolmogorov_smirnov(values, S, 1000)
```

```
The test parameter is: 0.02796970616465888
The test value is: 0.05146923119690054
This distribution passes the Kolmogorov and Smirnov Test
```

In [6]:
```
# Plotting the pseudo-random numbers generated and the sample distribution function to visualize the shape
plt.figure(1)
plt.hist(values, color='skyblue', edgecolor='black')
# Adding title, x label, and y label
plt.title('Histogram of pseudo-random numbers generated')
plt.xlabel('Pseudo-random numbers')
plt.ylabel('Frequency')
plt.show()

# Plotting the relationship between the sampling distribution function and the random numbers generated
plt.figure(2)
plt.plot(values, S, color='seagreen', label='Empirical Distribution Function')
plt.plot(values, values, color='red', label='CDF')
plt.legend()
# Adding title, x label, and y label
plt.title('EDF and CDF')
plt.xlabel('Pseudo-random values')
plt.ylabel('Distribution function')
plt.show()
```





In [7]:
```
# Using Monte-Carlo technique to approximate the expected value of our distribution
mean = sum(values)/1000
print("The estimated mean of the distribution is: " + str(mean))

# Using Monte-Carlo technique to approximate the variance of our distribution
values_sq = []
for i in range(len(values)):
    values_sq.append(values[i]**2)
variance = sum(values_sq)/1000 - mean**2
print("The estimated variance of the distribution is: " + str(variance))

# Calculating the approximate standard deviation of our distribution
std_dev = math.sqrt(variance)
print("The estimated standard deviation of the distribution is: " + str(std_dev))
```

```
The estimated mean of the distribution is: 0.4979613793538705
The estimated variance of the distribution is: 0.07874879765754572
The estimated standard deviation of the distribution is: 0.28062216173628507
```

Testing on Multiple Samples

In [8]:
```
# Defining a function that will perform the Kolmogorov and Smirnov test with suppressed output unless failed
def ks_suppressed(values, S, size):
    test_value = 1.6276/math.sqrt(size)
    diff = []
    for i in range(size):
        diff.append(abs(S[i] - values[i]))
    test_param = max(diff)

    if(test_param < test_value):
        return 1
    else:
        print("The test failed")
        print("The test parameter is: " + str(test_param))
        print("The test value is: " + str(test_value))
        return 0
```

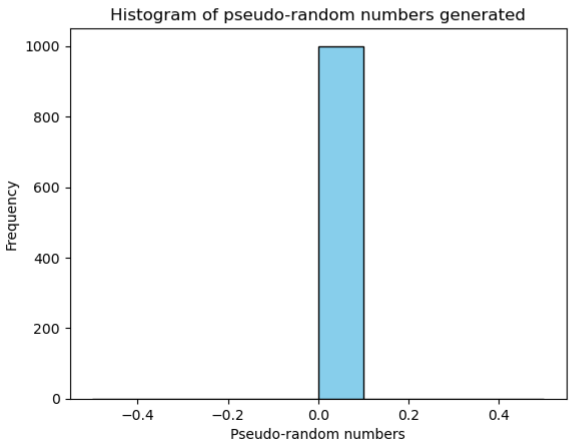
In [9]:
```
def generator_test(num_trials):
    # Generating multiple seeds to see if we still pass the test
    starting_values = []
    for i in range(0,num_trials):
        starting_values.append(i)

    # Running the test for each starting value and evaluating if it passes the test
    total_pass = 0
    for i in range(len(starting_values)):
        seed = starting_values[i]
        # Running the test
        values = pseudo_random_uniform(seed, 1000)
        S = sample_distribution(values, 1000)
        passed = ks_suppressed(values, S, 1000)
```

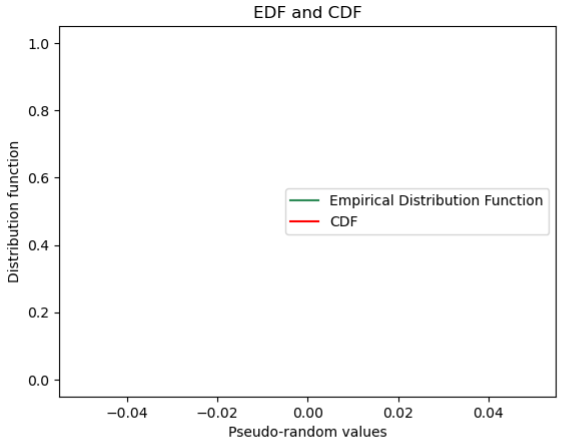
```
        if(passed == 1):
            total_pass += 1
        else:
            print("The seed that produced failing results: " + str(seed))

            print("For the pseudo-random numbers, the lower bound is " + str(min(values)) + " the upper bound is " + str(max

            # Using Monte-Carlo technique to approximate the expected value of our distribution
            mean = sum(values)/1000
            print("The estimated mean of the distribution is: " + str(mean))

            # Using Monte-Carlo technique to approximate the variance of our distribution
            values_sq = []
            for i in range(len(values)):
                values_sq.append(values[i]**2)
            variance = sum(values_sq)/1000 - mean**2
            print("The estimated variance of the distribution is: " + str(variance))

            # Calculating the approximate standard deviation of our distribution
            std_dev = math.sqrt(variance)
            print("The estimated standard deviation of the distribution is: " + str(std_dev))

            # Plotting the pseudo-random numbers generated and the sample distribution function to visualize the shape
            plt.figure(1)
            plt.hist(values, color='skyblue', edgecolor='black')
            # Adding title, x label, and y label
            plt.title('Histogram of pseudo-random numbers generated')
            plt.xlabel('Pseudo-random numbers')
            plt.ylabel('Frequency')
            plt.show()
```

```
            # Plotting the relationship between the sampling distribution function and the random numbers generated
            plt.figure(2)
            plt.plot(values, S, color='seagreen', label='Empirical Distribution Function')
            plt.plot(values, values, color='red', label='CDF')
            plt.legend()
            # Adding title, x label, and y label
            plt.title('EDF and CDF')
            plt.xlabel('Pseudo-random values')
            plt.ylabel('Distribution function')
            plt.show()

    print("The generator passes the test " + str(total_pass/len(starting_values)*100) + "% of the time with " + str(num_tria
```

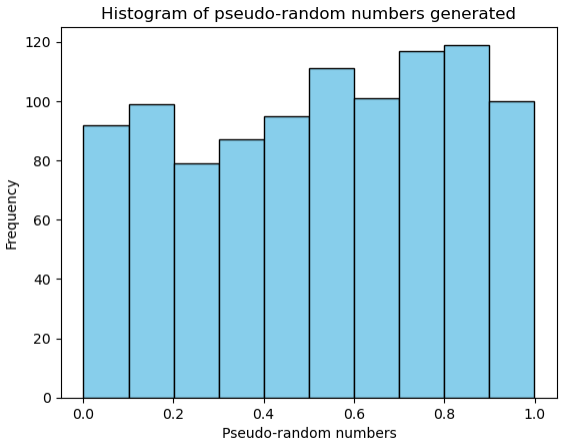
In [10]:
```
generator_test(1000)
```

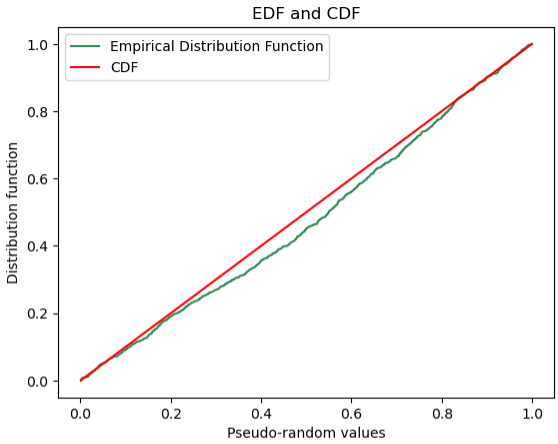
```
The test failed
The test parameter is: 1.0
The test value is: 0.05146923119690054
The seed that produced failing results: 0
For the pseudo-random numbers, the lower bound is 0.0 the upper bound is 0.0
The estimated mean of the distribution is: 0.0
The estimated variance of the distribution is: 0.0
The estimated standard deviation of the distribution is: 0.0
```



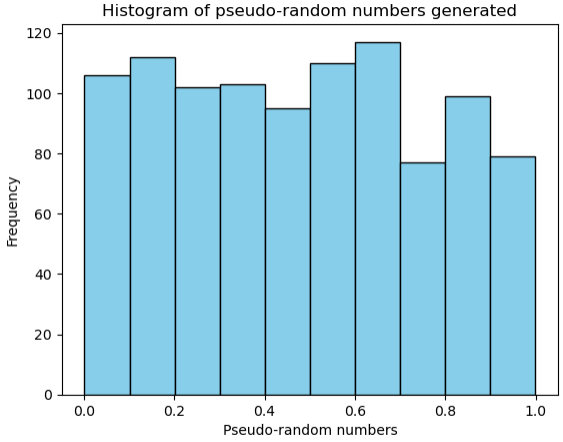


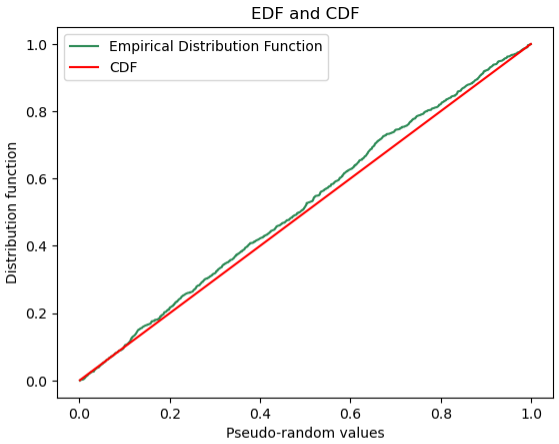

```
The test failed
The test parameter is: 0.05850634546974043
The test value is: 0.05146923119690054
The seed that produced failing results: 256
For the pseudo-random numbers, the lower bound is 0.001062598545599076 the upper bound is 0.9996327063998359
The estimated mean of the distribution is: 0.5231131145908093
The estimated variance of the distribution is: 0.0820583450296381
The estimated standard deviation of the distribution is: 0.28645827799112056
```



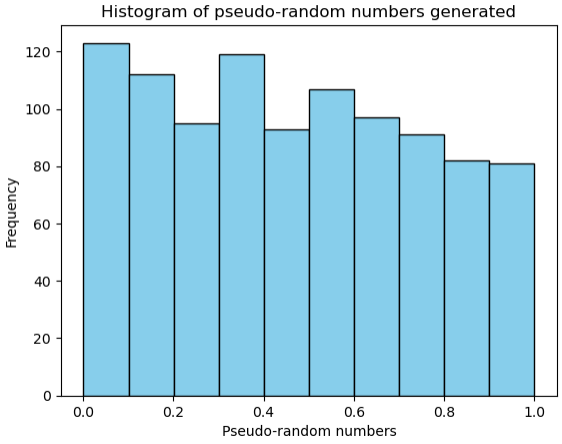


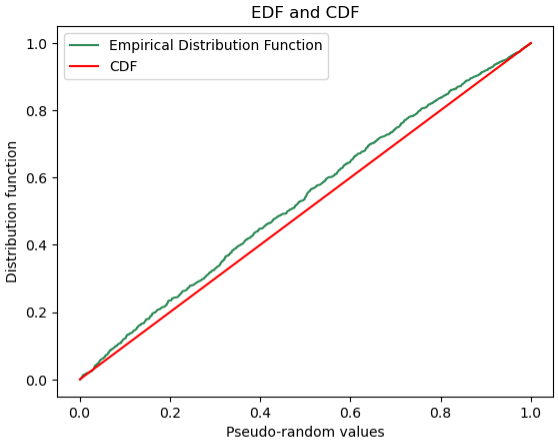

```
The test failed
The test parameter is: 0.0531549992855429
The test value is: 0.05146923119690054
The seed that produced failing results: 366
For the pseudo-random numbers, the lower bound is 0.0012143375357679732 the upper bound is 0.999609091784623
The estimated mean of the distribution is: 0.4798648435165477
The estimated variance of the distribution is: 0.08021696178315366
The estimated standard deviation of the distribution is: 0.2832259906561431
```



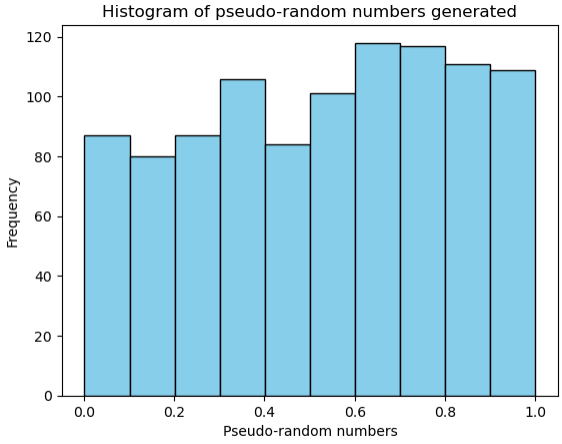


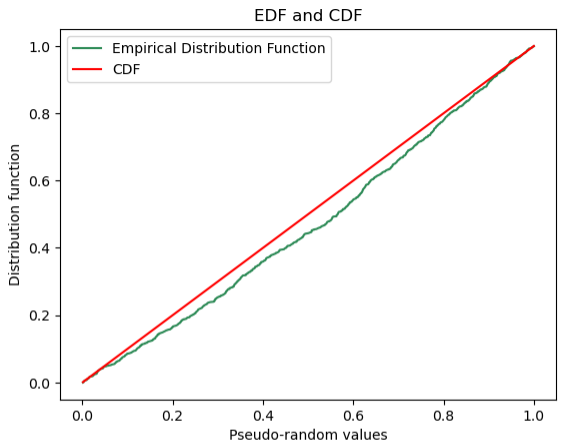

```
The test failed
The test parameter is: 0.05551308081881745
The test value is: 0.05146923119690054
The seed that produced failing results: 401
For the pseudo-random numbers, the lower bound is 0.0007768748331707319 the upper bound is 0.9998202007263062
The estimated mean of the distribution is: 0.4665131209020096
The estimated variance of the distribution is: 0.08154534697469637
The estimated standard deviation of the distribution is: 0.28556145918995507
```



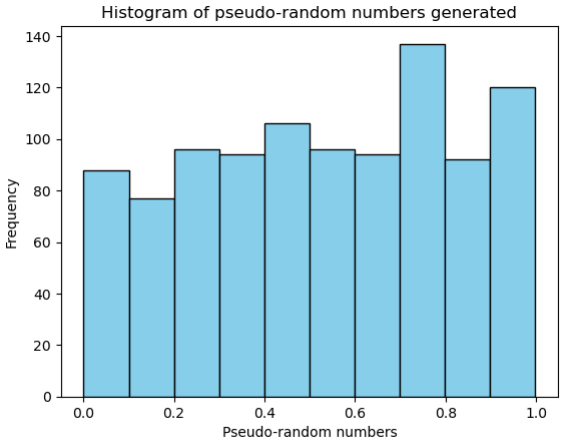


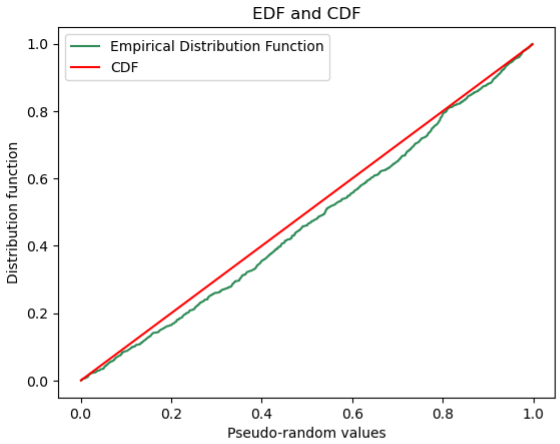

```
The test failed
The test parameter is: 0.06911511918209262
The test value is: 0.05146923119690054
The seed that produced failing results: 577
For the pseudo-random numbers, the lower bound is 0.0014562052681372525 the upper bound is 0.999773803632601
The estimated mean of the distribution is: 0.5317158871831908
The estimated variance of the distribution is: 0.08074227053832228
The estimated standard deviation of the distribution is: 0.28415184415787675
```



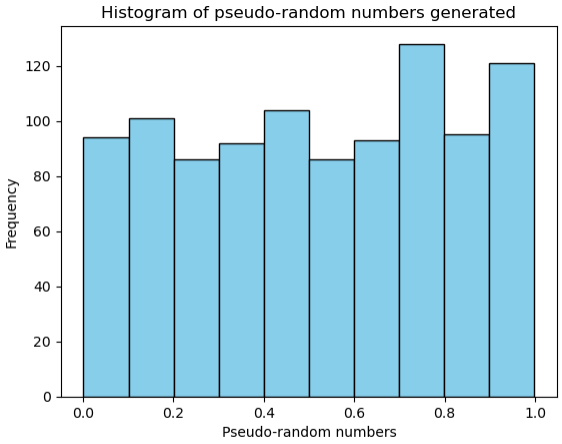


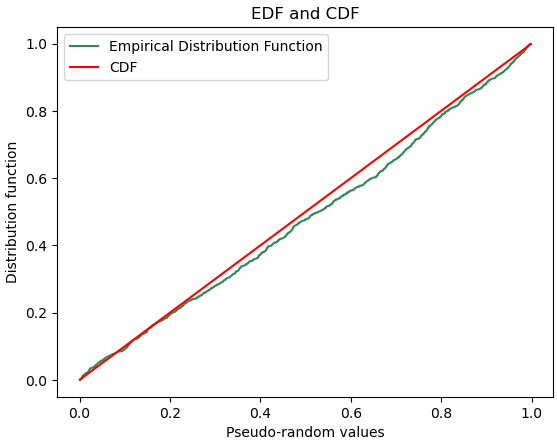

```
The test failed
The test parameter is: 0.053291065734481
The test value is: 0.05146923119690054
The seed that produced failing results: 598
For the pseudo-random numbers, the lower bound is 0.0007667220201188335 the upper bound is 0.99788354383683
The estimated mean of the distribution is: 0.5289048536144684
The estimated variance of the distribution is: 0.08171809114882228
The estimated standard deviation of the distribution is: 0.28586376326638935
```






```
The test failed
The test parameter is: 0.05274489378172198
The test value is: 0.05146923119690054
The seed that produced failing results: 931
For the pseudo-random numbers, the lower bound is 0.0008551981304097912 the upper bound is 0.9984027761958553
The estimated mean of the distribution is: 0.5190441784532944
The estimated variance of the distribution is: 0.08689703926850334
The estimated standard deviation of the distribution is: 0.2947830376200492
```






```
The generator passes the test 99.3% of the time with 1000 different seeds.
```