

**Національний Технічний Університет України КПІ**

Факультет Інформатики та Обчислювальної Техніки  
Кафедра інформатики та програмної інженерії

**Практична робота №1**

З дисципліни «Моделювання систем»

ПЕРЕВІРКА ГЕНЕРАТОРА ВИПАДКОВИХ ЧИСЕЛ НА ВІДПОВІДНІСТЬ ЗАКОНУ РОЗПОДІЛУ

**Перевірив:**

Асистент

Дифучин А.Ю.

Оцінка:\_\_\_\_\_\_\_\_\_\_\_**Виконав:**

Студент групи ІТ-04

Чапча С.О.

Дата: \_\_\_\_\_\_\_

**Завдання до практичної роботи**

* Згенерувати 10000 випадкових чисел трьома вказаними нижче способами.
* Згенерувати випадкове число за формулою , де ξi  - випадкове число, рівномірно розподілене в інтервалі (0;1). Числа ξi можна створювати за допомогою вбудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність експоненційному закону розподілу . Перевірку зробити при різних значеннях λ.
* Згенерувати випадкове число по формулах:

,

де ξi  - випадкове число, рівномірно розподілене в інтервалі (0;1). Числа ξi можна створювати за допомогою убудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність нормальному закону розподілу:

.

Перевірку зробити при різних значеннях *а* і σ.

* Згенерувати випадкове число за формулою , де *a*=513, *с*=231. Перевірити на відповідність рівномірному закону розподілу в інтервалі (0;1). Перевірку зробити при різних значеннях параметрів *а* і *с*.
* Для кожного побудованого генератора випадкових чисел побудувати гістограму частот, знайти середнє і дисперсію цих випадкових чисел. По виду гістограми частот визначити вид закону розподілу.
* Відповідність заданому закону розподілу перевірити за допомогою критерію згоди χ2.
* Зробити висновки щодо запропонованих способів генерування випадкових величин.

**Лістинг програми:**

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import chi2

from scipy.stats import expon

from scipy.stats import norm

from scipy.stats import uniform

lamArr = [0.5, 1.0, 1.5, 2.0, 2.5]

numOf = 10000

def xsquareExpTest(rndNumbers, numBins, alpha, lam, xMax=10):

oFrequencies = np.histogram(rndNumbers, bins=numBins, density=True)[0]

x = np.linspace(0, xMax, 50)

eFrequency = expon.pdf(x, scale=1/lam)

xsqrStats = np.sum((oFrequencies - eFrequency) \*\* 2 / eFrequency)

freedomDeg = numBins - 1

xsqrCrit = chi2.ppf(1 - alpha, freedomDeg)

return xsqrStats, xsqrCrit

for lam in lamArr:

rndNumbers = np.random.rand(numOf)

rndENumbers = - (1 / lam) \* np.log(rndNumbers)

plt.hist(rndENumbers, bins=50, density=True, alpha=0.6, label=f'λ = {lam}', color='blue')

xMax = np.max(rndENumbers)

x = np.linspace(0, xMax, 50)

expLaw = expon.pdf(x, scale=1/lam)

plt.plot(x, expLaw, 'r-', label='Exponential Law')

average = np.mean(rndENumbers)

dispersion = np.var(rndENumbers)

xsqrStats, xsqrCrit = xsquareExpTest(rndENumbers, 50, 0.05, lam, xMax)

print(f'λ is {lam}:')

print(f'Average: {average}')

print(f'Dispersion: {dispersion}')

print(f'XSquare: [{xsqrStats}, {xsqrCrit}]')

if xsqrCrit >= xsqrStats:

print('Distribution Match')

else:

print('Distibution Doesnt Match')

plt.legend()

plt.title(f'Histogram for λ = {lam}')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.show()

numOf = 10000

aArr = [0, 2, 4]

omegaArr = [1, 2, 3]

def xsquareTest(rndNumbers, numBins, alpha):

oFrequencies = np.histogram(rndNumbers, bins=numBins, density=True)[0]

eFrequency = norm.pdf(np.linspace(-3, 3, numBins))

xsqrStats = np.sum((oFrequencies - eFrequency) \*\* 2 / eFrequency)

freedomDeg = numBins - 1

xsqrCrit = chi2.ppf(1 - alpha, freedomDeg)

return xsqrStats, xsqrCrit

def calculateUi():

xiArr = np.random.rand(12)

return np.sum(xiArr - 6)

def generateXArr(omega, a):

mu = np.array([calculateUi() for \_ in range(numOf)])

return omega \* mu + a

for a in aArr:

for omega in omegaArr:

xArr = generateXArr(omega, a)

average = np.mean(xArr)

dispersion = np.var(xArr)

xRange = np.linspace(min(xArr), max(xArr), 100)

normal = norm.pdf(xRange, loc=average, scale=np.sqrt(dispersion))

plt.hist(xArr, bins=30, density=True, alpha=0.6, label=f'omega:[{omega}] a:[{a}]', color='blue')

plt.plot(xRange, normal, 'r-', label=f'avg:[{average}] sigma:[{np.sqrt(dispersion)}]')

xsqrStats, xsqrCrit = xsquareTest(xArr, 30, 0.05)

print(f'Omega is {omega} and A is {a}')

print(f'Dispersion is {dispersion}')

print(f'Average is {average}')

print(f'XSquare: [{xsqrStats}, {xsqrCrit}]')

if xsqrCrit >= xsqrStats:

print('Distribution Match')

else:

print('Distibution Doesnt Match')

plt.legend()

plt.title(f'Histogram for omega={omega} and a={a}')

plt.show()

numOf = 10000

aArr = [5\*\*13, 5\*\*5, 2\*\*20]

cArr = [2\*\*31, 2\*\*13, 5\*\*20]

def generateNumbers(a, c, numOf):

z = np.zeros(numOf)

x = np.zeros(numOf)

z[0] = 1

for i in range(1, numOf):

z[i] = (a \* z[i-1]) % c

x[i] = (z[i] + c) / c

return x

def xsquareUniTest(rndNumbers, numBins, alpha):

oFrequencies = np.histogram(rndNumbers, bins=numBins, density=True)[0]

eFrequency = uniform.pdf(np.linspace(0, 1, numBins))

xsqrStats = np.sum((oFrequencies - eFrequency) \*\* 2 / eFrequency)

freedomDeg = numBins - 1

xsqrCrit = chi2.ppf(1 - alpha, freedomDeg)

return xsqrStats, xsqrCrit

for a in aArr:

for c in cArr:

rndNumbers = generateNumbers(a, c, numOf)

plt.hist(rndNumbers, bins=50, density=True, alpha=0.6, color='blue', label='Histogram Of Frequencies')

average = np.mean(rndNumbers)

dispersion = np.var(rndNumbers)

xsqrStats, xsqrCrit = xsquareUniTest(rndNumbers, 50, 0.05)

uni = uniform.pdf(np.linspace(0, 1, 50))

plt.plot(np.linspace(1, 2, 50), uni, 'r-', label='Distribution Density')

print(f'a is {a} and c is {c}')

print(f'Average: {average}')

print(f'Dispersion: {dispersion}')

print(f'XSquare: [{xsqrStats}, {xsqrCrit}]')

if xsqrCrit >= xsqrStats:

print('Distribution Match')

else:

print('Distibution Doesnt Match')

plt.xlim(1, 2)

plt.legend()

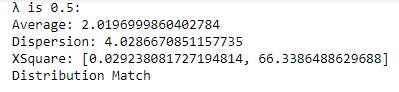
plt.xlabel('Random Numbers')

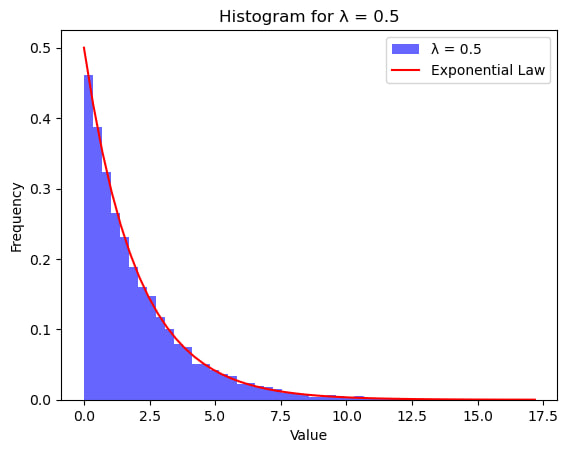
plt.ylabel('Frequency')

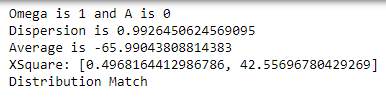
plt.title(f'Histogram for a={a} and c={c}')

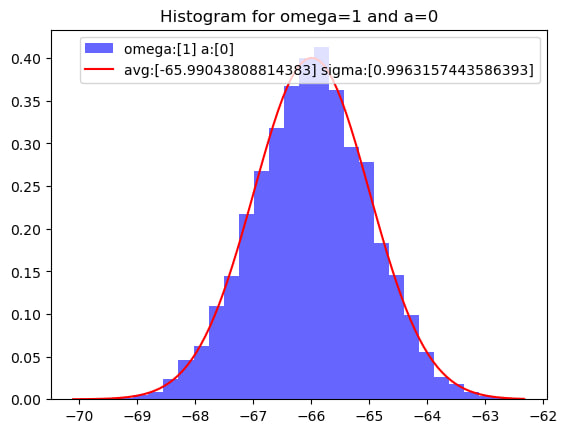
plt.show()

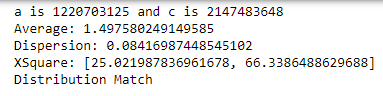
**Результати роботи програми:**

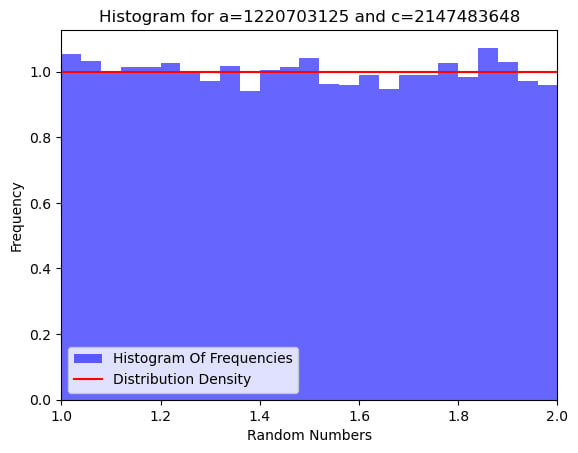












**Висновок:**

Після виконання перевірок генераторів випадкових чисел на відповідність різним законам розподілу, хочеться зауважити, що величина своїх довільних констант, які наявні у формулах генерації випадкових величин, можуть суттєво вплинути на якість отриманого розподілу або ж на величину критерію Пірсона, за допомогою якого ми перевіряємо відповідність заданому закону розподілу.

Посилання на гітхаб: <https://github.com/whitetark/modelling-systems>