НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО”

КАФЕДРА АВТОМАТИЗОВАНИХ СИСТЕМ ОБРОБКИ ІНФОРМАЦІЇ І УПРАВЛІННЯ

Комп’ютерний практикум № 1

з дисципліни

“Моделювання систем”

Виконала:

студентка групи ІС-71

Алпаєва Ю.С

Перевірила:

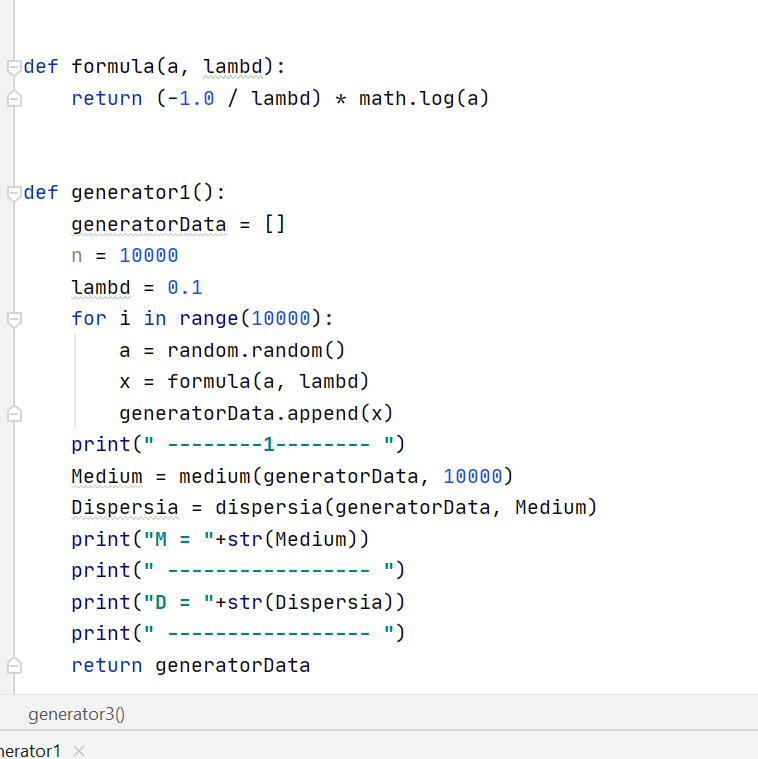
старший викладач

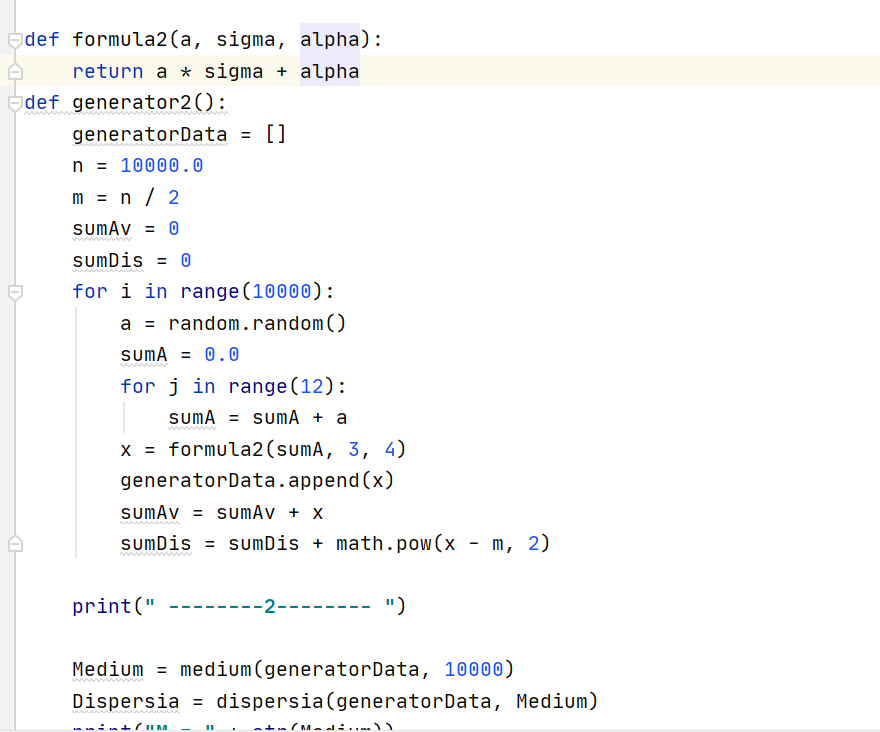
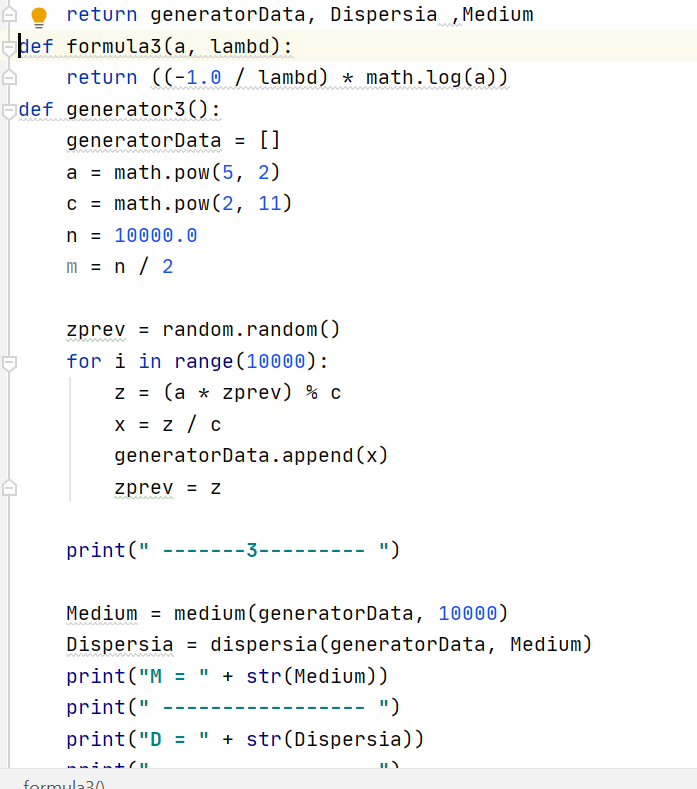
Новікова П.А.

Комп’ютерний практикум №1

«Перевірка генератора випадкових чисел на відповість закону розподілу»

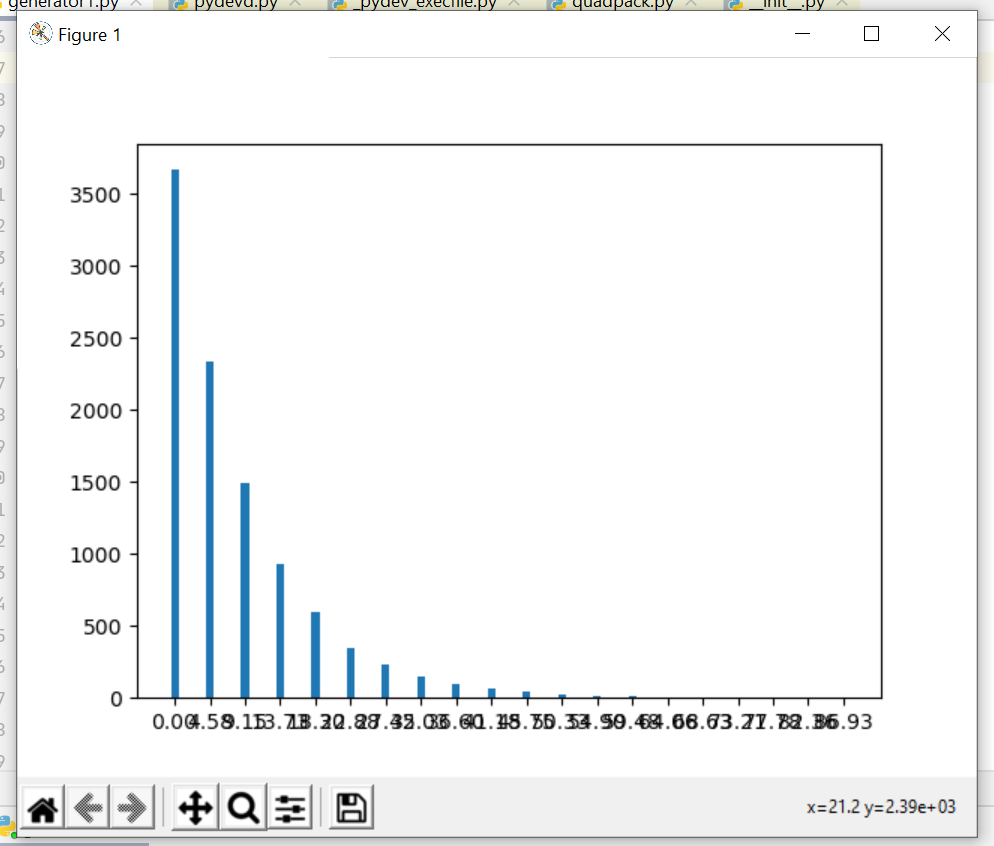
Згенерувати 10000 випадкових чисел трьома вказаними нижче способами. **45 балів**.

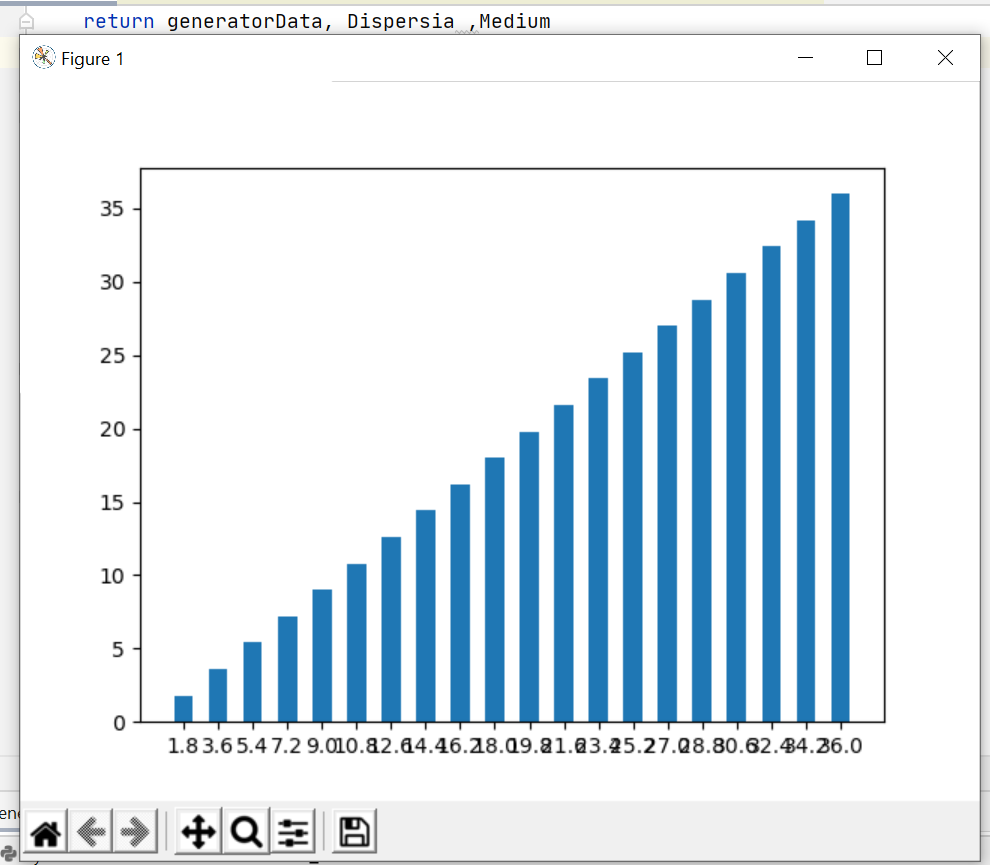


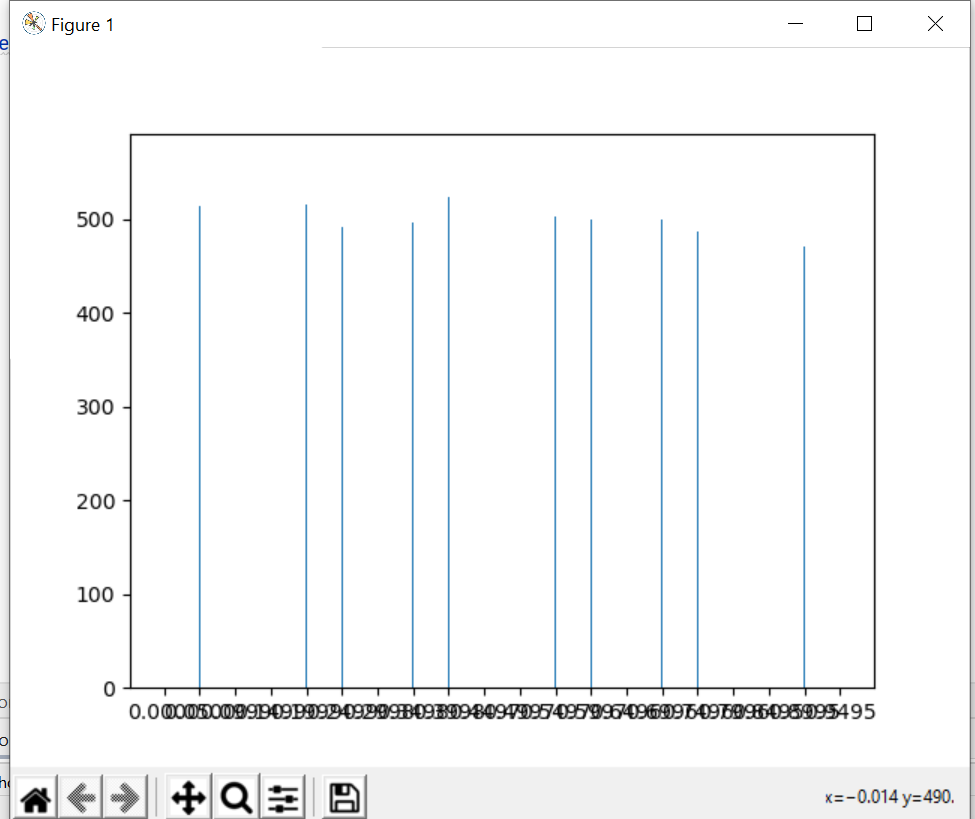
Для кожного побудованого генератора випадкових чисел побудувати гістограму частот

Перший генератор



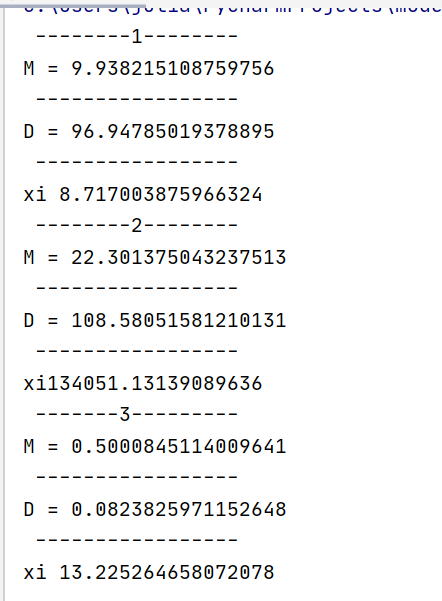
Другий генератор 

Третій генератор



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. **30 балів**



Код програми

import math  
import random  
import matplotlib.pyplot as plt  
import numpy as np  
from scipy import integrate  
  
  
def formula(a, lambd):  
 return (-1.0 / lambd) \* math.log(a)  
  
  
def generator1():  
 generatorData = []  
 n = 10000  
 lambd = 0.1  
 for i in range(10000):  
 a = random.random()  
 x = formula(a, lambd)  
 generatorData.append(x)  
 print(**" --------1-------- "**)  
 Medium = medium(generatorData, 10000)  
 Dispersia = dispersia(generatorData, Medium)  
 print(**"M = "**+str(Medium))  
 print(**" ----------------- "**)  
 print(**"D = "**+str(Dispersia))  
 print(**" ----------------- "**)  
 return generatorData  
  
  
def formula2(a, sigma, alpha):  
 return a \* sigma + alpha  
def generator2():  
 generatorData = []  
 n = 10000.0  
 m = n / 2  
 sumAv = 0  
 sumDis = 0  
 for i in range(10000):  
 a = random.random()  
 sumA = 0.0  
 for j in range(12):  
 sumA = sumA + a  
 x = formula2(sumA, 3, 4)  
 generatorData.append(x)  
 sumAv = sumAv + x  
 sumDis = sumDis + math.pow(x - m, 2)  
  
 print(**" --------2-------- "**)  
  
 Medium = medium(generatorData, 10000)  
 Dispersia = dispersia(generatorData, Medium)  
 print(**"M = "** + str(Medium))  
 print(**" ----------------- "**)  
 print(**"D = "** + str(Dispersia))  
 print(**" ----------------- "**)  
 return generatorData, Dispersia ,Medium  
def formula3(a, lambd):  
 return ((-1.0 / lambd) \* math.log(a))  
def generator3():  
 generatorData = []  
 a = math.pow(5, 2)  
 c = math.pow(2, 11)  
 n = 10000.0  
 m = n / 2  
  
 zprev = random.random()  
 for i in range(10000):  
 z = (a \* zprev) % c  
 x = z / c  
 generatorData.append(x)  
 zprev = z  
  
 print(**" -------3--------- "**)  
  
 Medium = medium(generatorData, 10000)  
 Dispersia = dispersia(generatorData, Medium)  
 print(**"M = "** + str(Medium))  
 print(**" ----------------- "**)  
 print(**"D = "** + str(Dispersia))  
 print(**" ----------------- "**)  
 return generatorData  
  
  
def findA(xi, genData=[]):  
 for j in range(len(genData)):  
 if (genData[j] > xi):  
 return genData[j - 1]  
 return genData[len(genData) - 1]  
  
  
def findB(xi1, genData=[]):  
 for j in range(len(genData)):  
 if (genData[j] > xi1):  
 return genData[j]  
 return genData[len(genData) - 1]  
  
  
def xiGenerator3(genData=[], x=[], y=[], max=0, min=0):  
 p = np.empty(len(x) - 1)  
 genData = np.sort(genData);  
 *# bpref = 0* d = 0;  
 *# for m in range(len(genData)):  
 # if (genData[m] > x[0]):  
 # b = genData[m]  
 # break* for i in range(len(x)):  
 if (i == len(x) - 1):  
 *# p[i] = (x[i]) / (max - min)* continue  
 else:  
 *# for j in range(len(genData)):  
 # if(genData[j-1]> x[i+1]):  
 # b= genData[j]  
 # a = bpref* p[d] = (findB(x[i + 1], genData) - findA(x[i], genData)) / (max - min)  
 d = d + 1;  
 *# bpref = b* return xiCriteria(p, y, 10000)  
  
  
def xiGenerator1(x=[], y=[], lambd=0.1):  
 p = np.empty(20)  
 for i in range(20):  
 if (i == 19):  
 continue  
 else:  
 p[i] = math.exp((-(lambd \* x[i]))) - math.exp((-(lambd \* x[i + 1])))  
 return xiCriteria(p, y, 10000)  
def combine(x = [],y = []):  
 newX =x  
 newY =y  
 for(i) in range(len(y)):  
 if(y[i]<5):  
 *#x.remove(i+1)* y[i+1] = y[i+1]+y[i]  
 *#y.remove(i)* for m in range (len(x)):  
 if(m == i+1):  
 continue  
 newX[m] =x[m]  
 for r in range (len(y)):  
 if(r == i):  
 continue  
 newY[r] =x[r]  
 return newX,newY  
def findLow(rand = [],a = 0,b = 0):  
 np.sort(rand);  
 for i in range(len(rand)):  
 if(rand[i]>a and rand[i]<b):  
 return rand[i]  
 return  
def xiGenerator2(x=[], y=[],rand = [],disp = 0,medium = 0):  
 p = np.empty(len(x)-1)  
 m = 0  
 x= np.sort(x)  
 rand = np.sort(rand)  
 x,y = combine(x,y)  
 f = lambda x: math.exp(-math.pow((x - medium), 2) / (2 \* disp))/math.sqrt(2 \* math.pi \* disp)  
 for i in range(1,len(x)):  
 if (i == (1)):  
 *#integral = integrate.quad(f, np.min(rand), x[i])* integral = simpson(f,np.min(rand),x[i],0.1)  
 else:  
 low =0  
 for j in range(len(rand)):  
 a = x[i-1]  
 b = x[i]  
 r = rand[j]  
 if (r > a and r < b):  
 low = r  
 break  
 *#integral =integrate.quad(f,low, x[i])* integral = simpson(f, low, x[i], 0.1)  
  
 p[m] =integral  
 m= m+1  
 return xiCriteria(p, y, 10000)  
  
  
def xiCriteria(p=[], ni=[], n=0):  
 xi = 0  
 for i in range(len(p)):  
 xi = xi + (math.pow(ni[i] - n \* p[i], 2)) / (n \* p[i])  
 return xi  
  
  
def draw1():  
 x = []  
 y = []  
 intervalsCount = 20  
 randomData = generator1()  
 h = (np.max(randomData) - np.min(randomData)) / intervalsCount  
 curh = 0  
 prefh = 0  
 m = 0  
 for i in range(20):  
 x.append(curh)  
 prefh = curh  
 curh = curh + h  
 m = 0  
 for j in range(len(randomData)):  
 if (randomData[j] > prefh and randomData[j] < curh):  
 m = m + 1  
 y.append(m)  
 print( **"xi "** + str(xiGenerator1(x, y)))  
 width = 1  
 plt.bar(x, y, width)  
 plt.xticks(x)  
 plt.show()  
  
def draw2():  
 x = []  
 y = []  
 intervalsCount = 20  
 randomData, D ,M = generator2()  
 h = (np.max(randomData) - np.min(randomData)) / intervalsCount  
 curh = h  
 m = 0  
 for i in range(20):  
 x.append(curh)  
 prefh = curh  
 curh = curh + h  
 m = 0  
 for j in range(len(randomData)):  
 if (randomData[j] > prefh and randomData[j] < curh):  
 m = m + 1  
 y.append(m)  
 print( **"xi"** +str( xiGenerator2(x, y,randomData, D ,M)))  
 width = 1  
 plt.bar(x, y, width)  
 plt.xticks(x)  
 plt.show()  
  
  
def draw3():  
 x = []  
 y = []  
 intervalsCount = 20  
 randomData = generator3()  
 h = (np.max(randomData) - np.min(randomData)) / intervalsCount  
 curh = 0  
 prefh = 0  
 m = 0  
 for i in range(20):  
 x.append(curh)  
 prefh = curh  
 curh = curh + h  
 m = 0  
 for j in range(len(randomData)):  
 if (randomData[j] > prefh and randomData[j] < curh):  
 m = m + 1  
 y.append(m)  
 newX = []  
 for j in range(len(x)):  
 if (x[j] > 0 and x[j] < 1):  
 newX.append(x[j])  
 print( **"xi "** + str(xiGenerator3(randomData, newX, y, np.max(randomData), np.min(randomData))))  
 width = 0.001  
 plt.bar(x, y, width)  
 plt.xticks(x)  
 plt.show()  
  
  
def dispersia(randomData=[], medium=0):  
 dispersiaValue = 0  
 for i in range(len(randomData)):  
 dispersiaI = math.pow(randomData[i] - medium, 2)  
 dispersiaValue = dispersiaValue + dispersiaI  
 return dispersiaValue / len(randomData)  
  
  
def medium(randomData=[], n=0):  
 return np.sum(randomData) / n  
  
  
*# def xiGenerator2(x=[], y=[], h=0, n=0, sigma=0,rand = []):  
# p = np.empty(len(x) - 1)  
# m = 0  
# h = (np.max(rand)- np.min(rand))/20  
# for i in range(20):  
# if (i == len(x) - 1):  
# continue  
# # if (i == 2):  
# # continue  
# # if (i == 1):  
# # continue  
# else:  
# z = (x[i + 1] - x[i]) / sigma  
# c1 = (h\*10) / sigma  
# c2 = 1 / (math.sqrt(2 \* math.pi))  
# p[m] = c1 \* math.pow(c2, math.exp(-math.pow(z, 2) / 2))  
# m = m + 1  
# return xiCriteria(p, y, 10000)*def f1 (x,Dispersia,Medium):  
 return 1 / math.sqrt(2 \* math.pi \* Dispersia) \* math.exp(-math.pow((x - Medium), 2) / (2 \* Dispersia));  
  
  
def simpson(f, a, b,n):  
 h=(b-a)/n  
 k=0.0  
 x=a + h  
 for i in range(1,100 + 1):  
 k += 4\*f(x)  
 x += 2\*h  
  
 x = a + 2\*h  
 for i in range(1,100):  
 k += 2\*f(x)  
 x += 2\*h  
 return (h/3)\*(f(a)+f(b)+k)  
draw1()  
draw2()  
draw3()

**Висновок**

В ході данної лабораторної роботи було розроблено три генератори випадкових числел, генератори використовували вбудовані засоби мови програмування для генераціі псевдорандомних числел, третій генератор використовував рекурсивну формулу.

Як бачимо критерій χ2  для генераторів 1 та 3 має прийнятне значення, а отже належність розподілу доведено.