Міністерство освіти і науки України

Національний авіаційний університет

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Імітаційне моделювання

Самостійна робота №2

«Дослідження генераторів випадкових чисел для різних законів розподілення»

Роботу виконав:
студент групи СП-325
Козлов Олексій
Роботу прийняла:
Нечипорук О.П.

Київ – 2020

Мета роботи: Ознайомитись із алгоритмами роботи генераторів випадкових чисел, що мають різні закони розподілення та методикою їх тестування.

1. Теоретичний опис функцій
   1.1. Створення файлу

   При виконані цієї функції формується масив випадкових величин за заданим законом розподілу (Тип розподілу), параметри розподілу також задані у полу "номер залікової книжки", обсяг вибірки задається в полі "обсяг вибірки", після створення вибірки вона зберігається у форматі JSON у файлі що вказаний в полі "шлях до файлу", якщо файл існує, то він перезаписується, якщо ні, то створюється.

   1.2. Тип величини

   При виконанні цієї функції виконується перевірка типу випадкової величини, чи є вона рівномірною чи дискретною. Спочатку масив випадкових величин читається із файлу, що вказаний в полі "шлях до фалу", потім оцінюється обсяг вибірки, і кількість унікальних значень в цій вибірці, в разі якщо вони співпадають, випадкова величина є рівномірною, в іншому випадку, вона дискретна. Метод перевірки працює доки обсяг вибірки не перевищує періодичність генераторів випадкових величин.

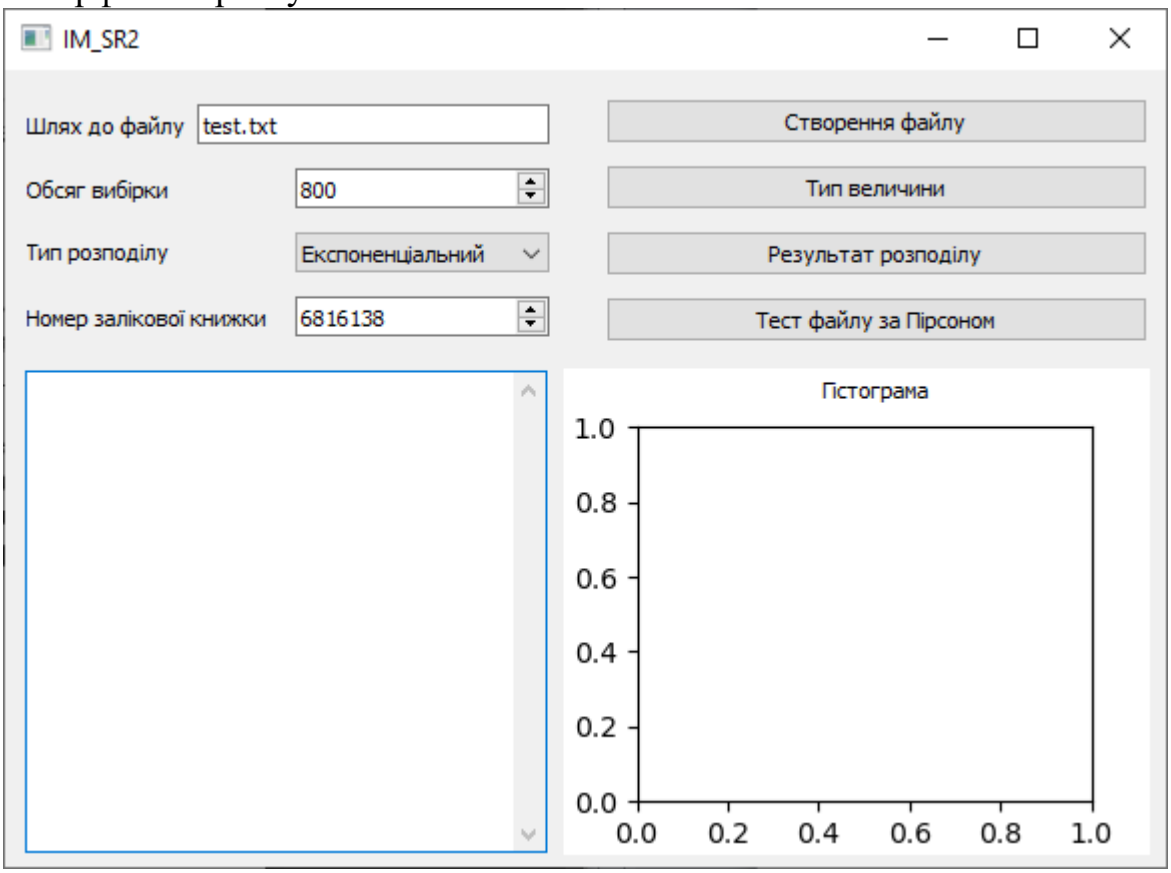
   1.3. Результат розподілу

   При виконанні цієї функції виводиться вся вибірка випадкових величин та гістограма цієї вибірки.

   1.4. Тест за критерієм Пірсона

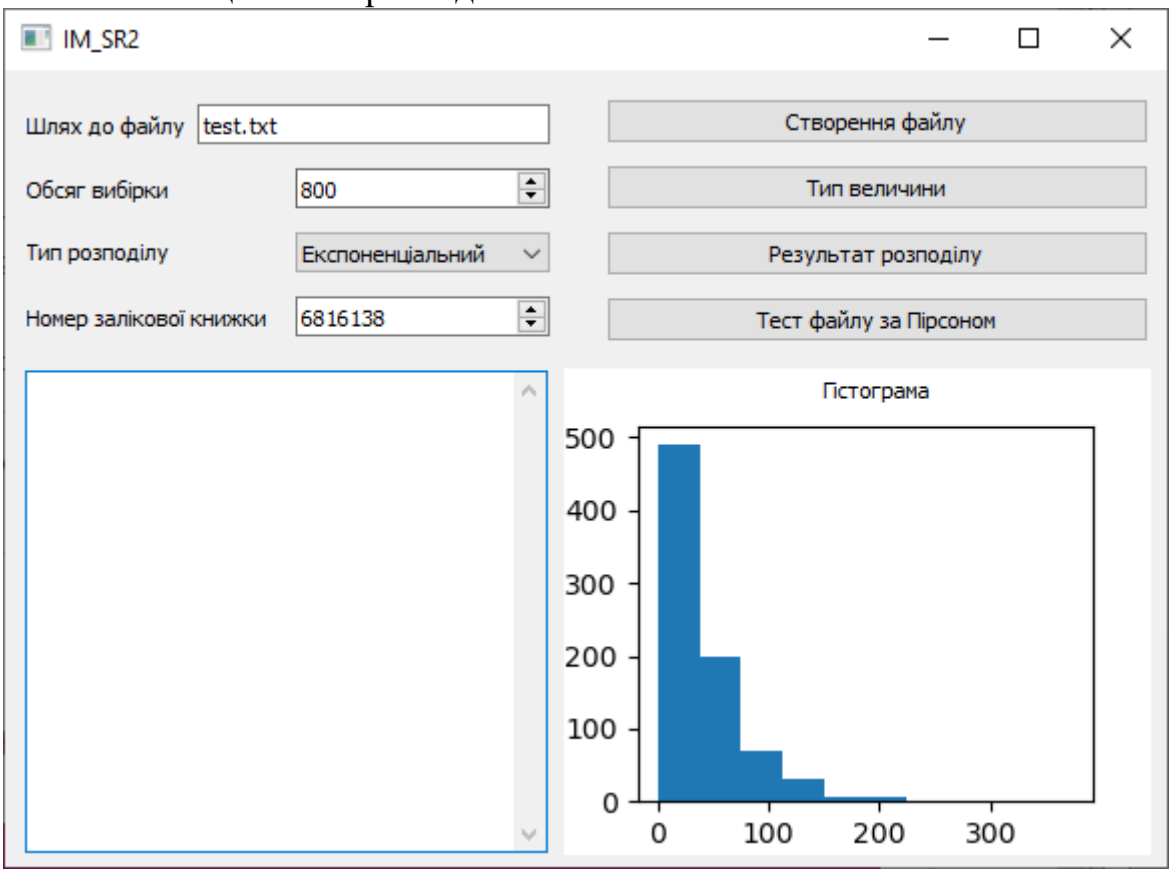
   При виконанні цієї функції виконується перевірка статистичної гіпотези на рівномірність за критерієм Пірсона. Для всіх типів розподілу окрім рівномірного Хі-квадрат має значно перевищувати критичне значення, Для рівномірного розподілу Хі-квадрат має бути меншим за критичне значення.

2. Інтерфейс користувача

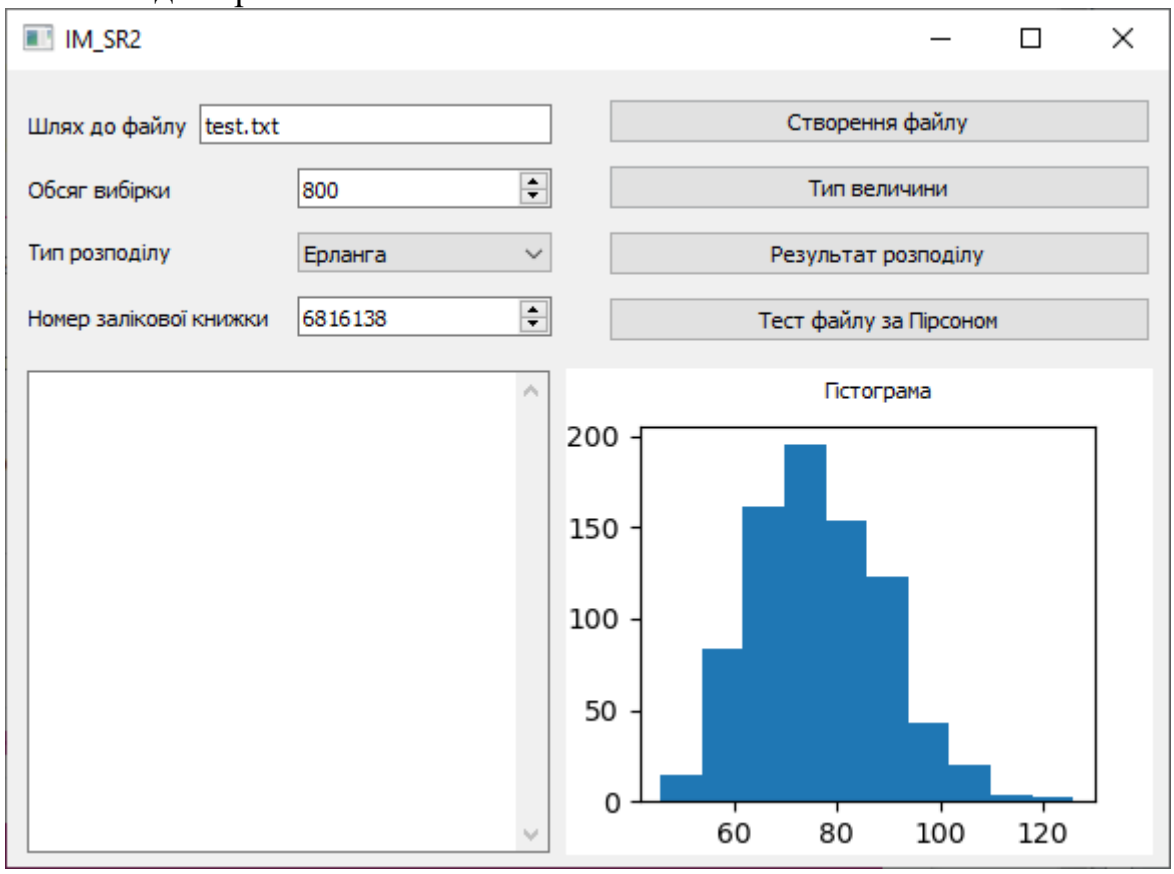


3. Генерація випадкових чисел

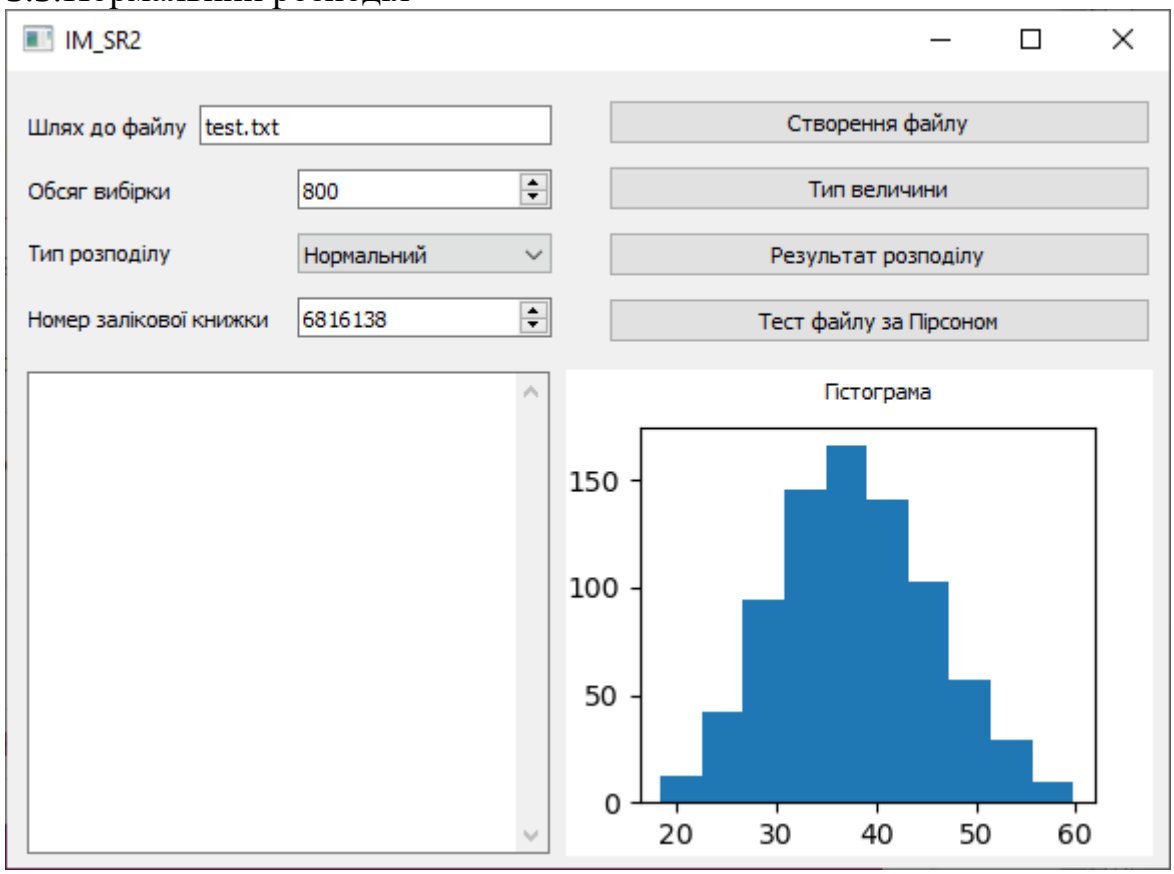
3.1.Експоненціальний розподіл



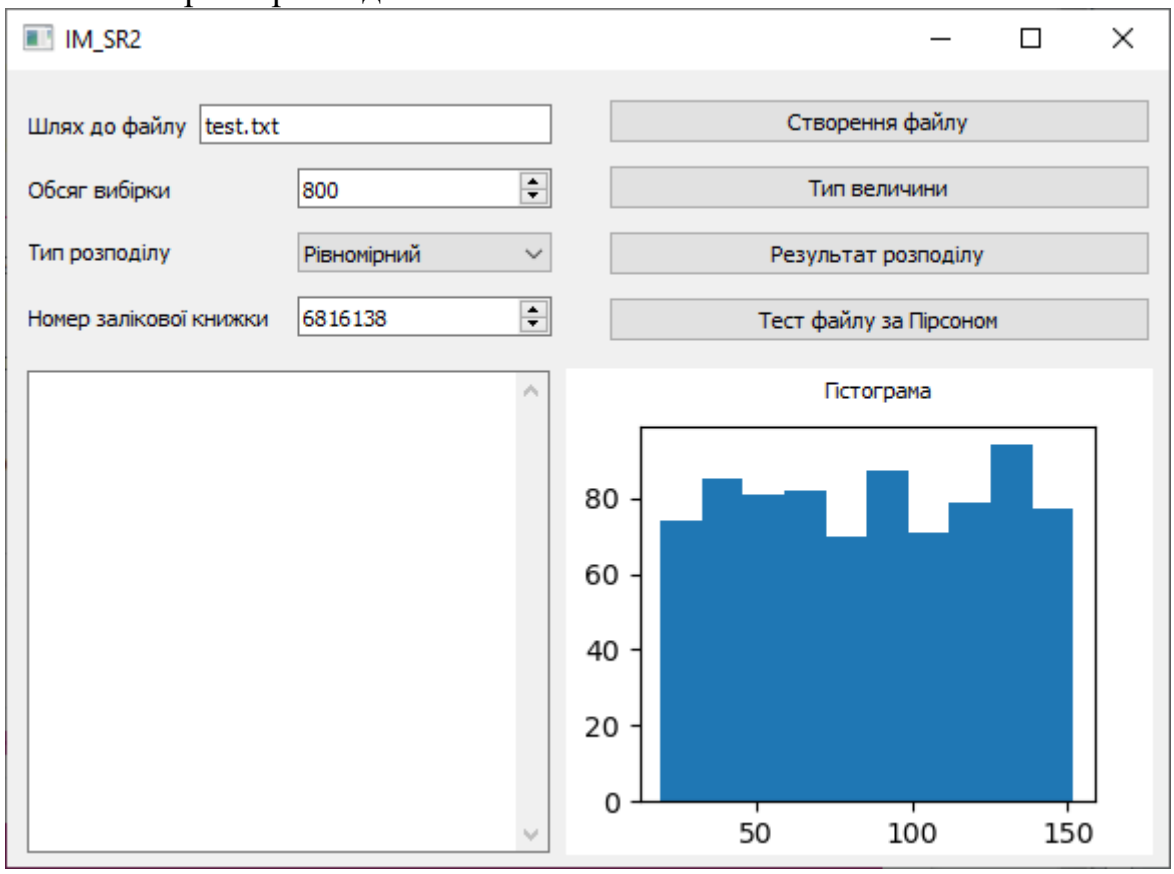
3.2.Розподіл Ерланга



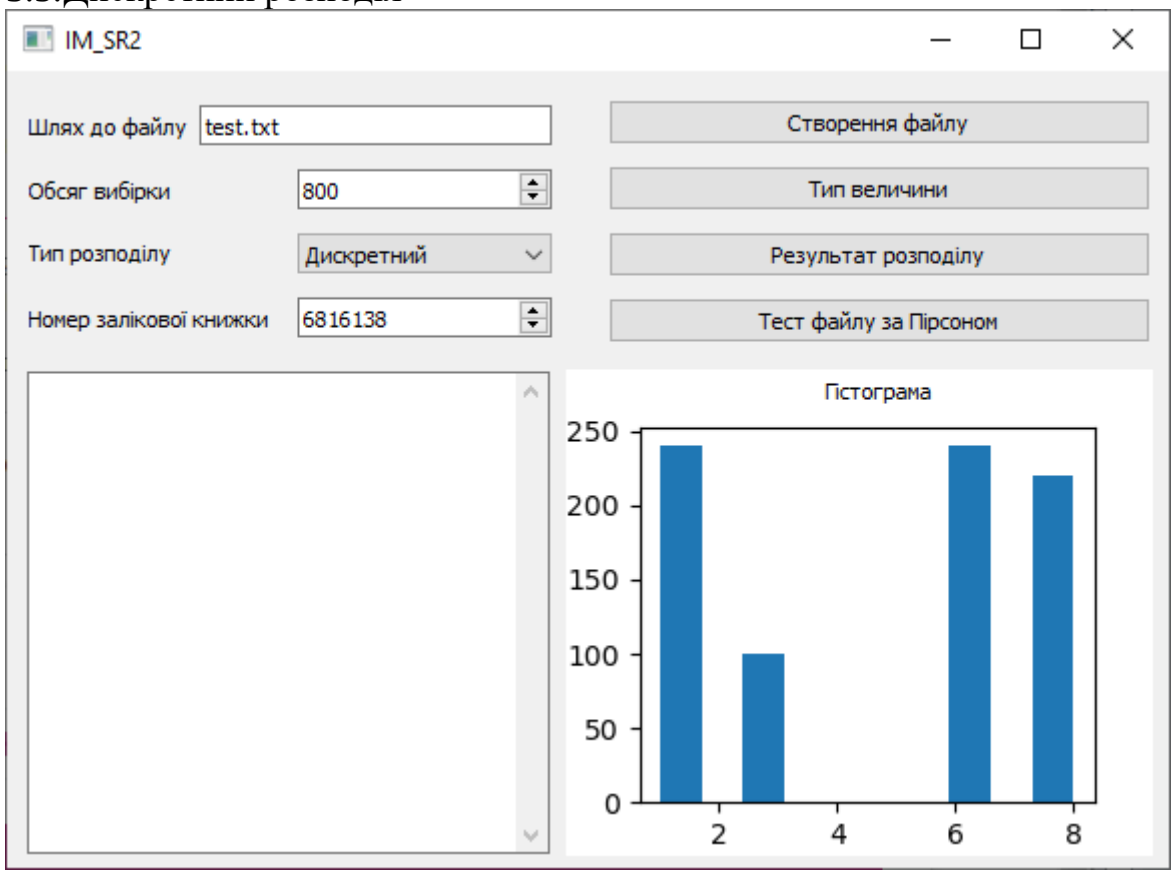
3.3.Нормальний розподіл



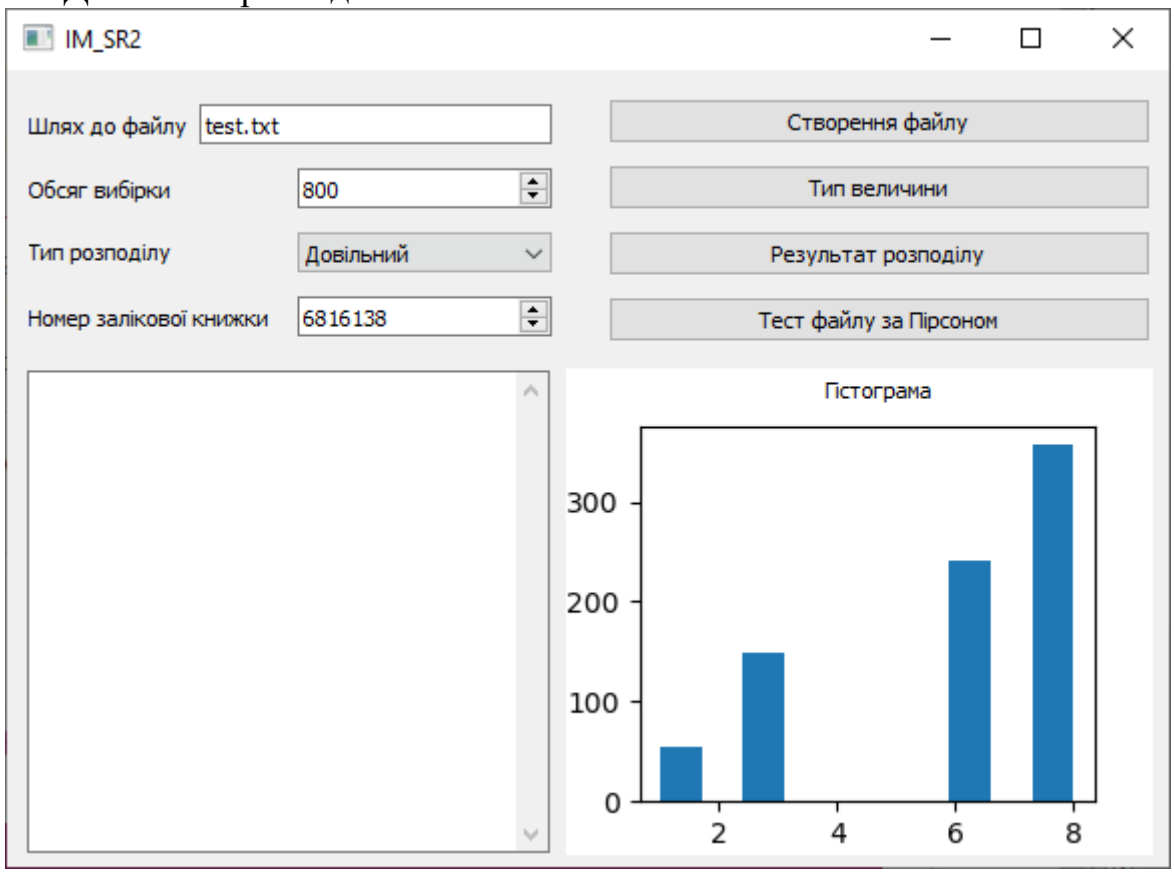
3.4.Рівномірний розподіл



3.5.Дискретний розподіл

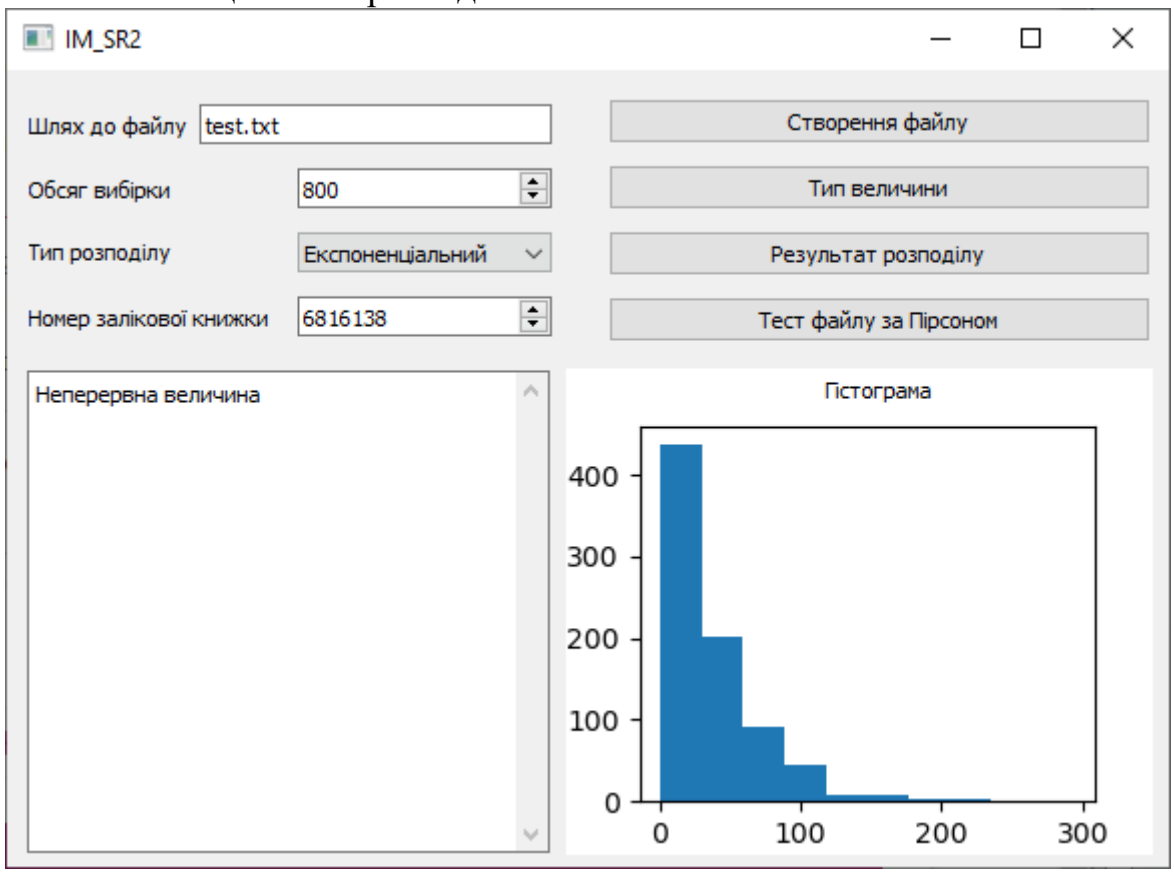


3.6.Довільний розподіл

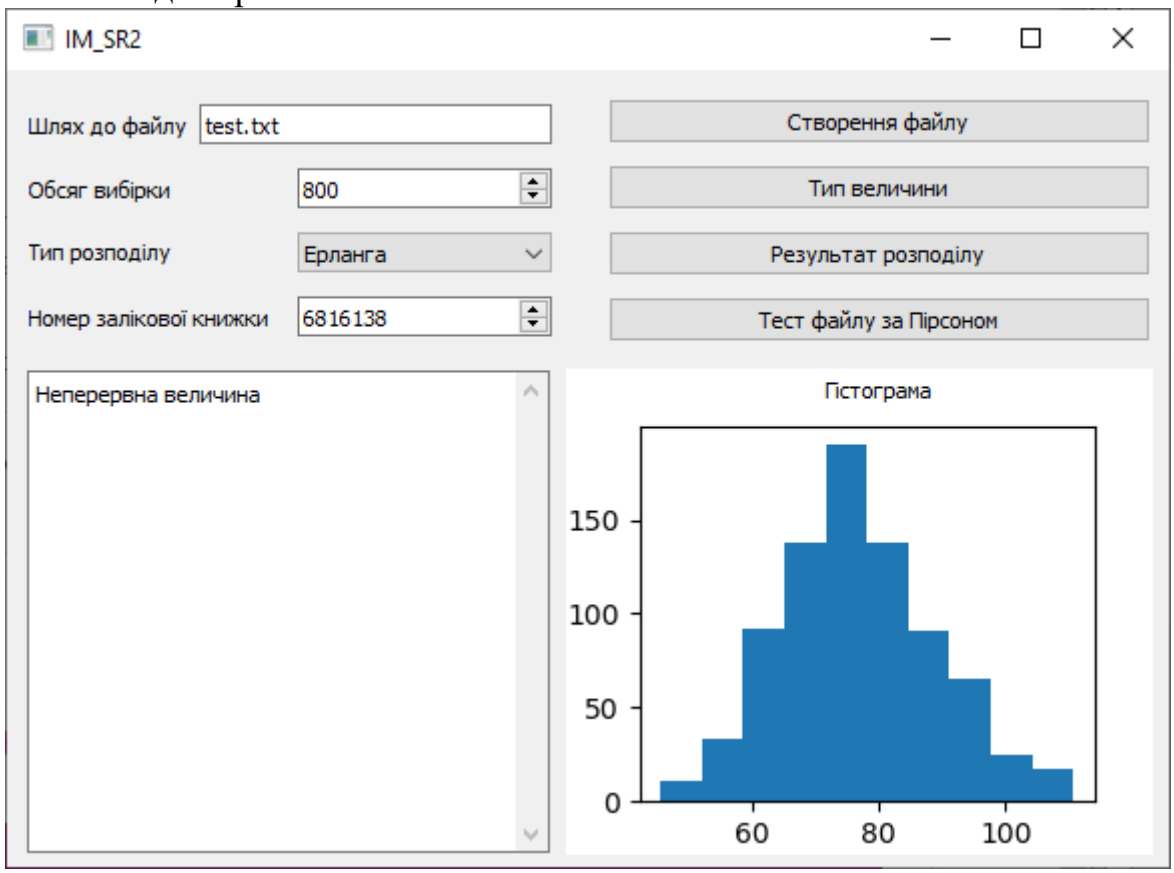


4. Перевірка типу величини

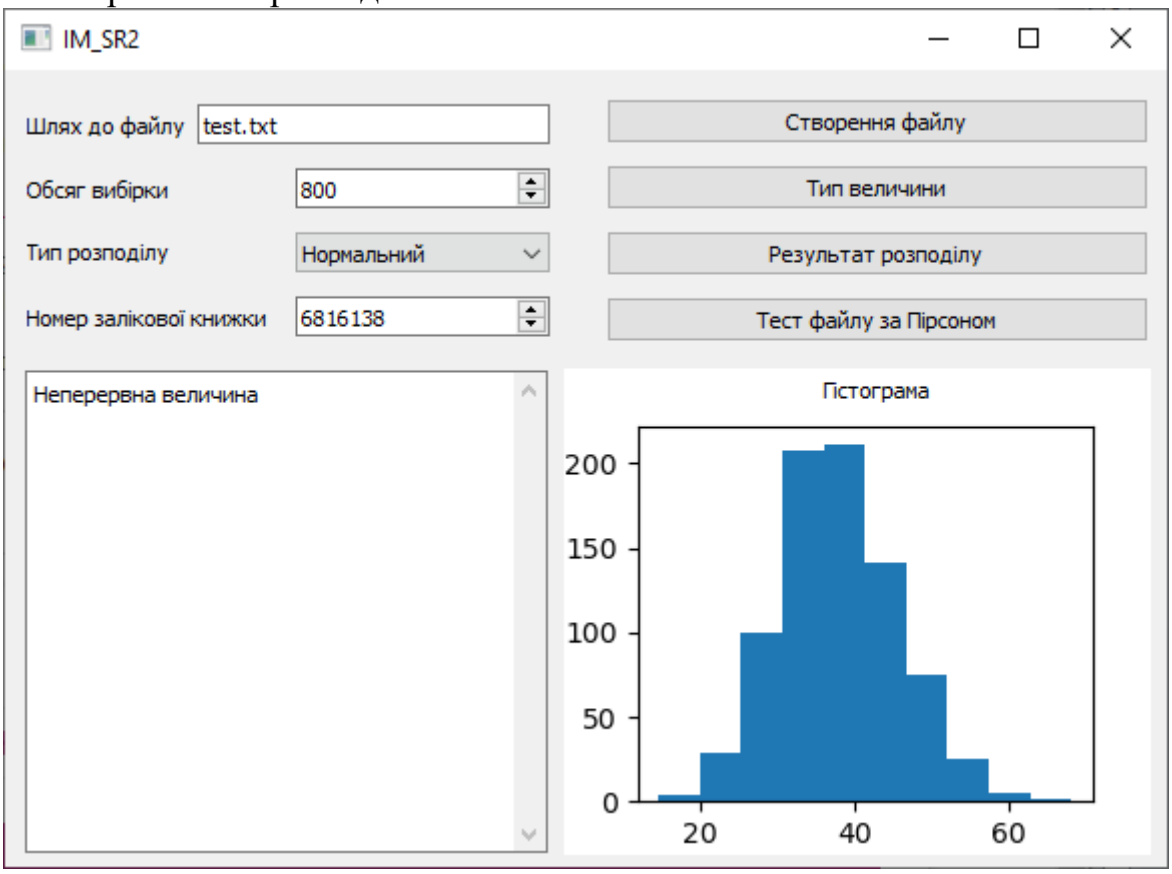
4.1.Експоненціальний розподіл



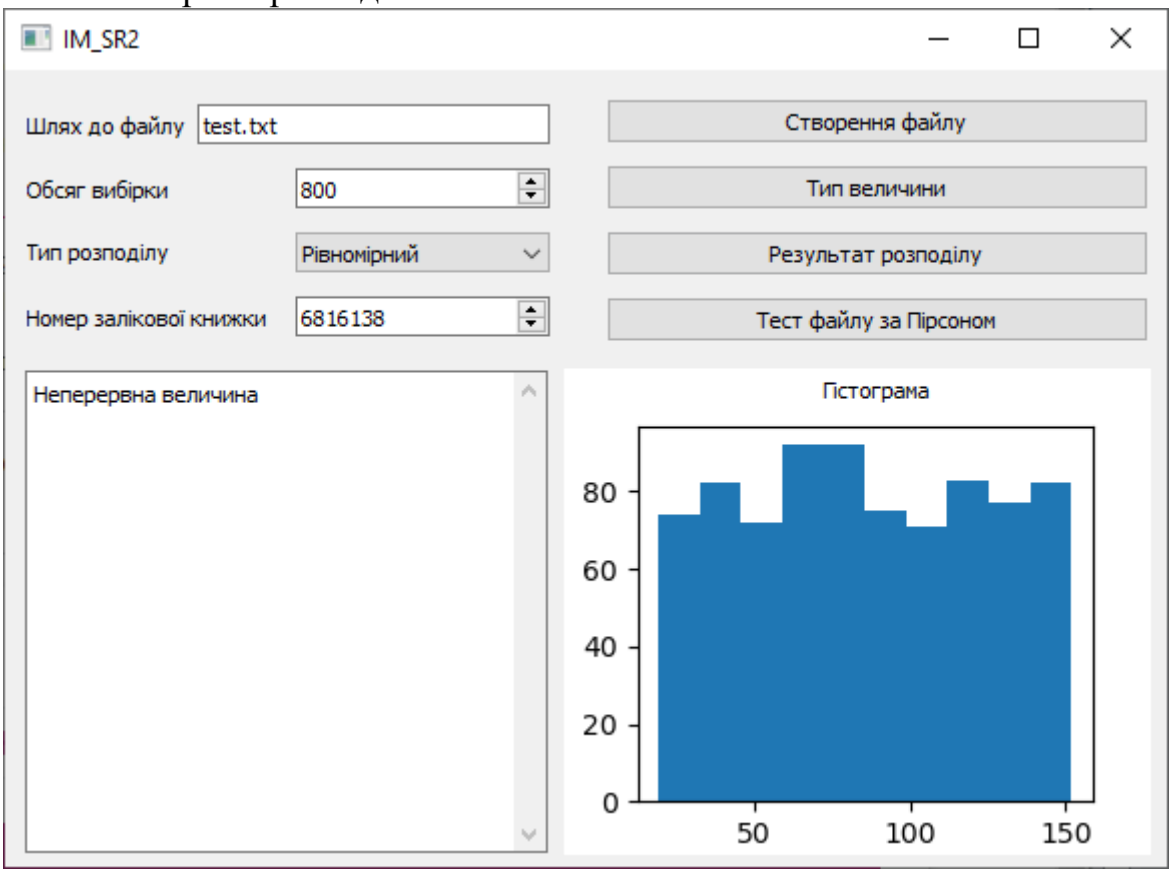
4.2.Розподіл Ерланга



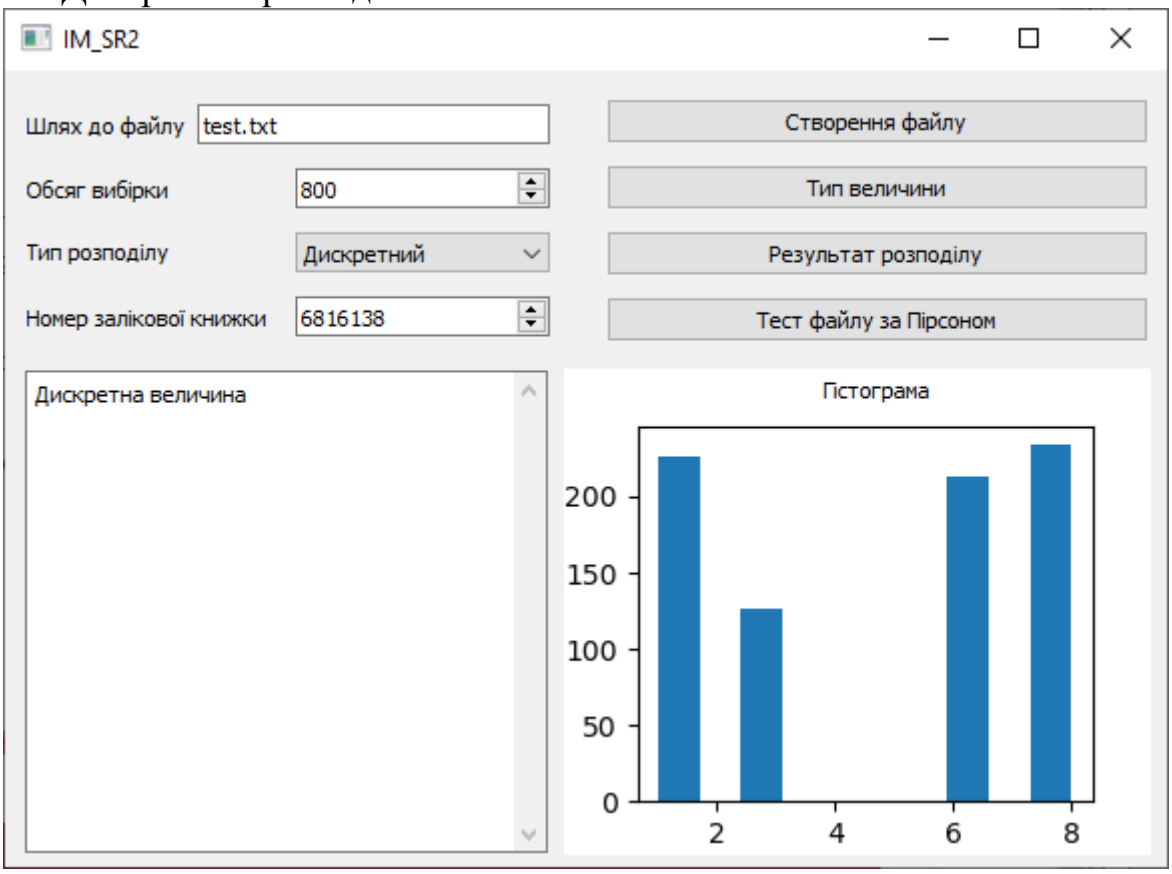
4.3.Нормальний розподіл



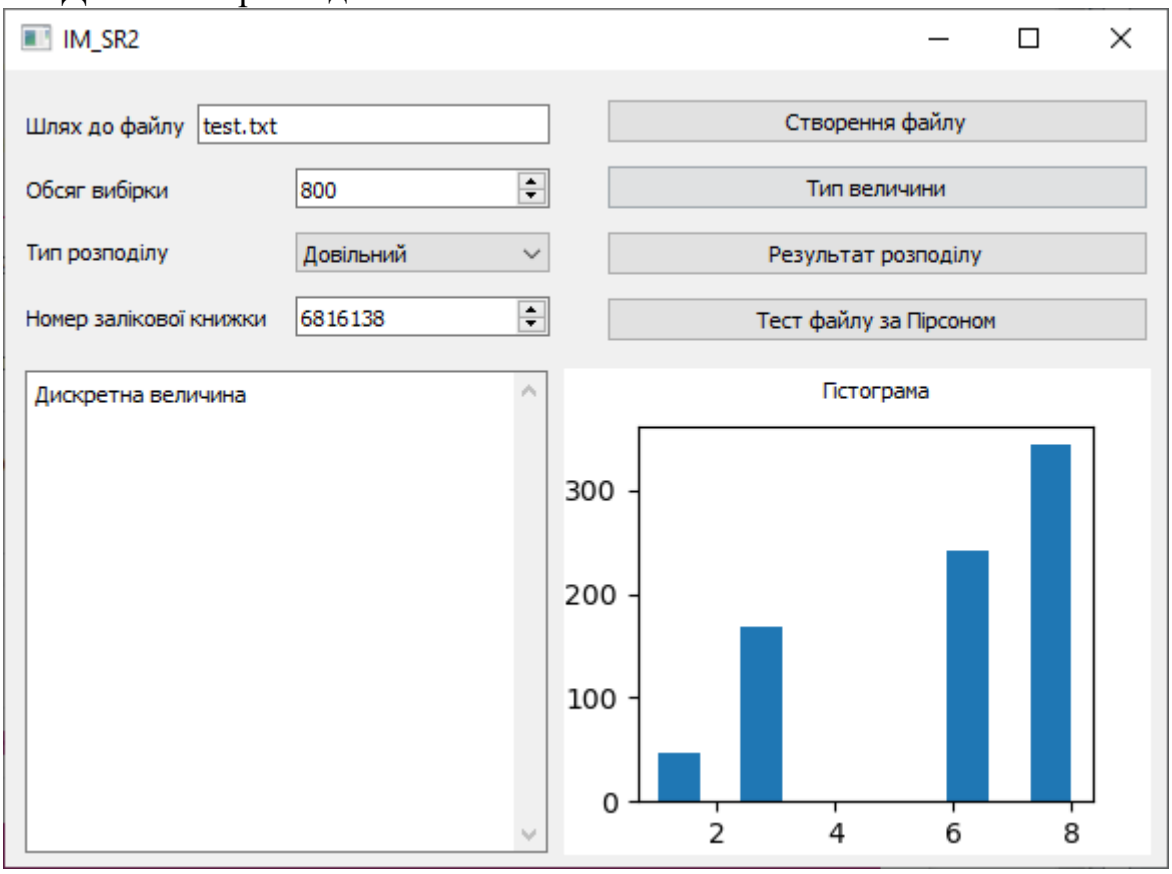
4.4.Рівномірний розподіл



4.5.Дискретний розподіл

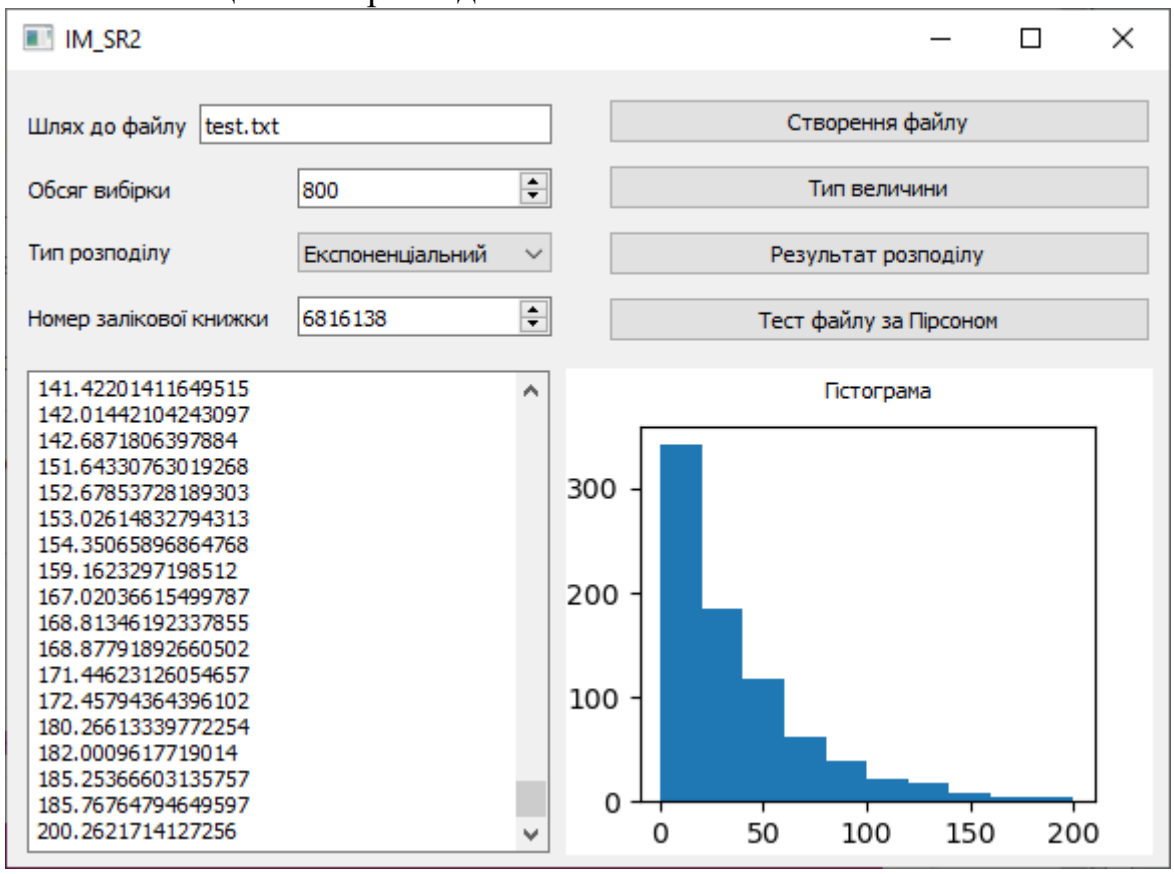


4.6.Довільний розподіл

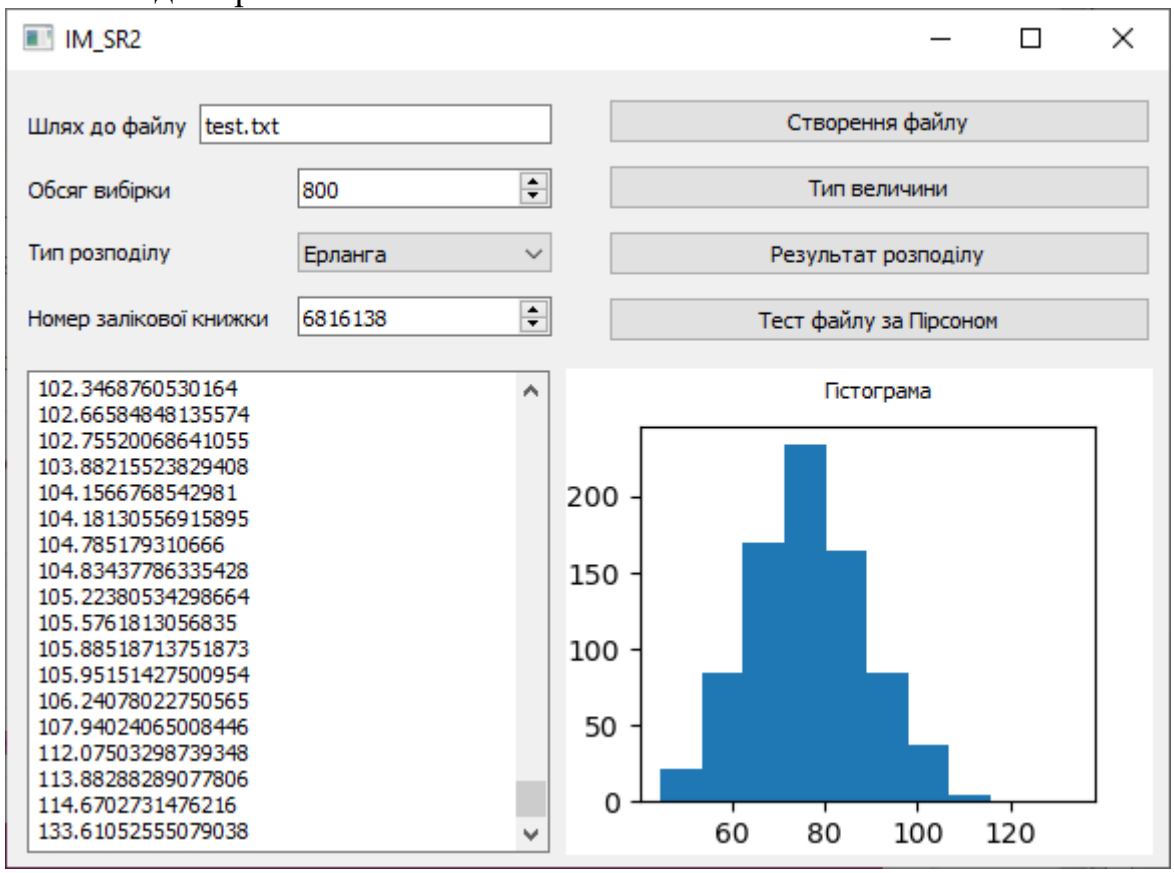


5. Результат розподілу

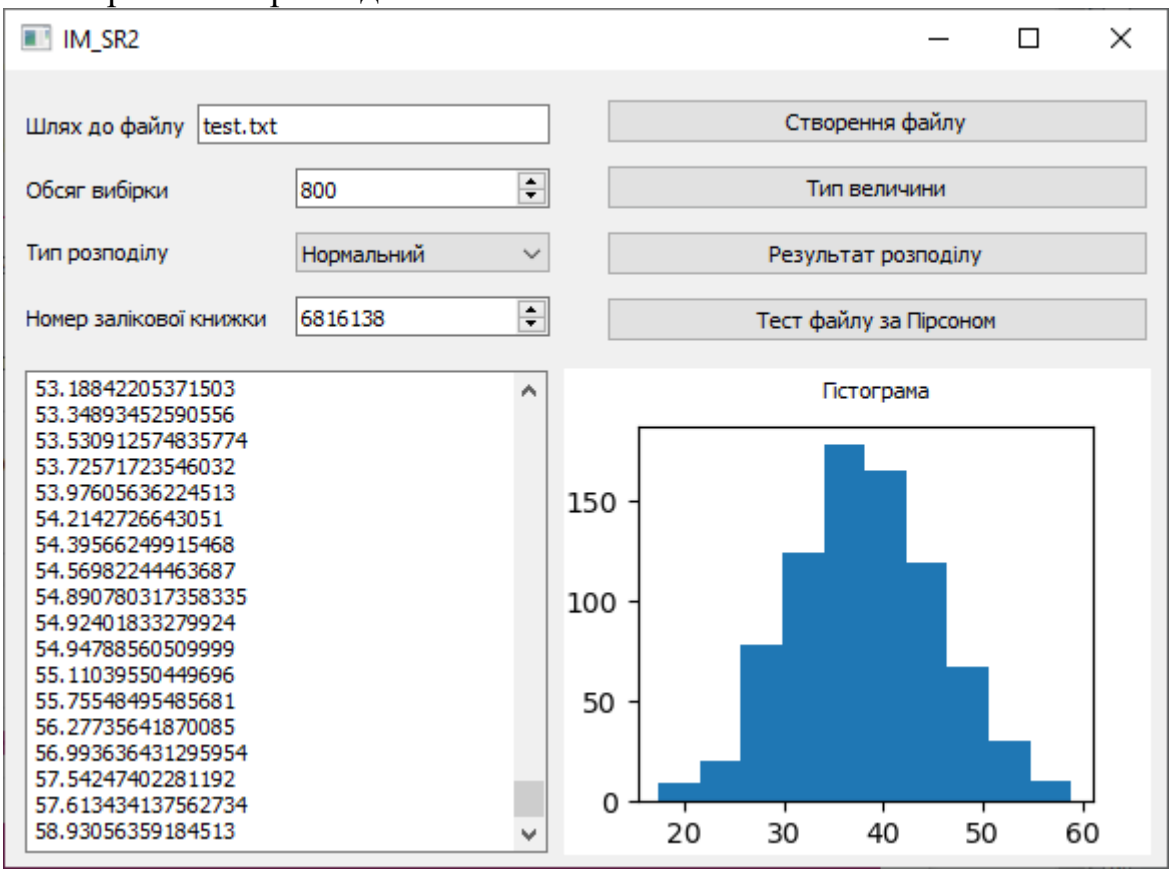
5.1.Експоненціальний розподіл



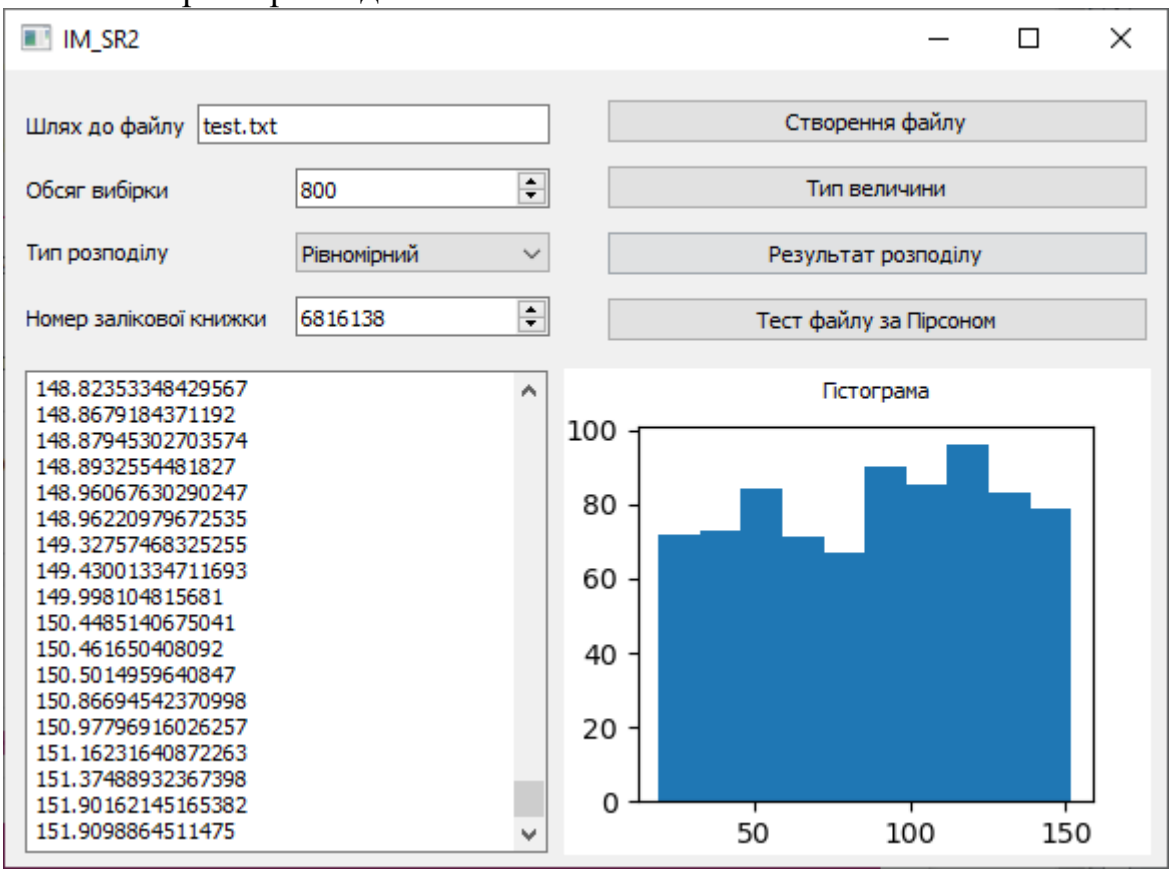
5.2.Розподіл Ерланга



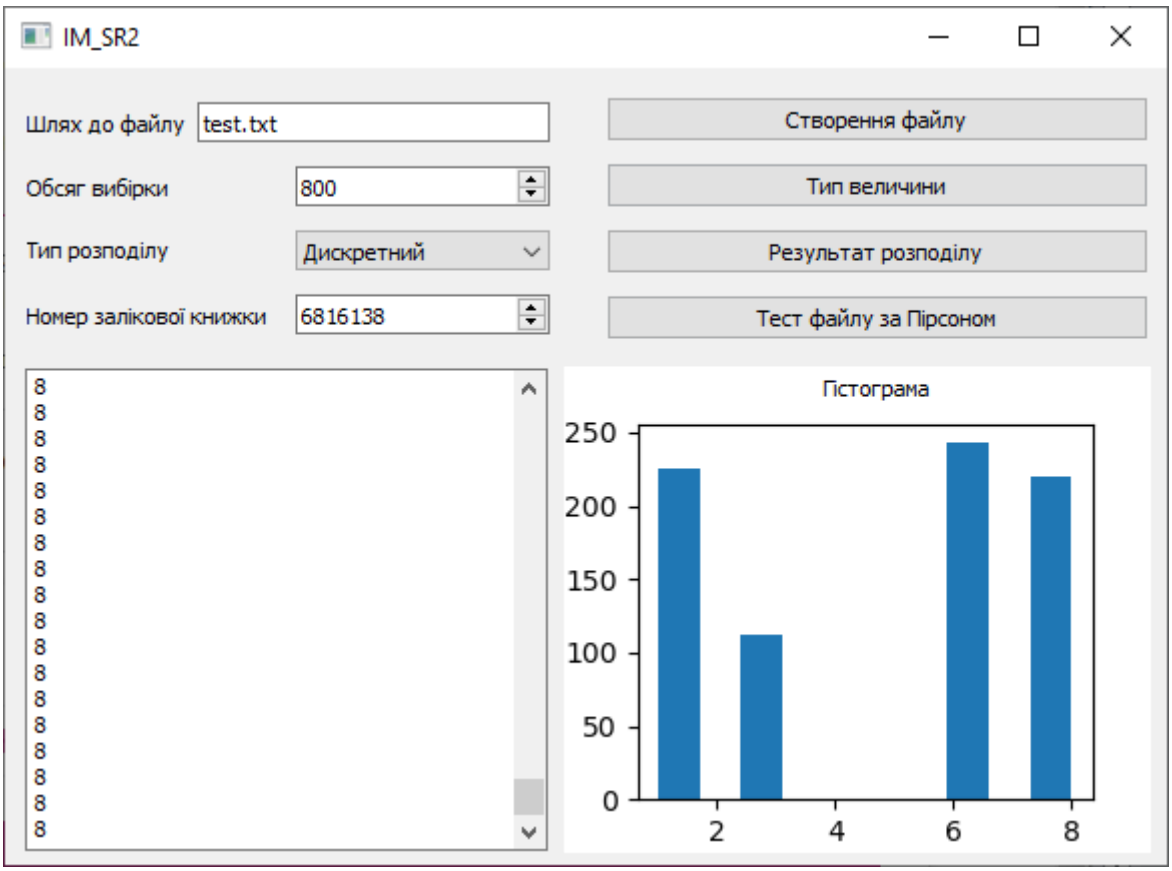
5.3.Нормальний розподіл



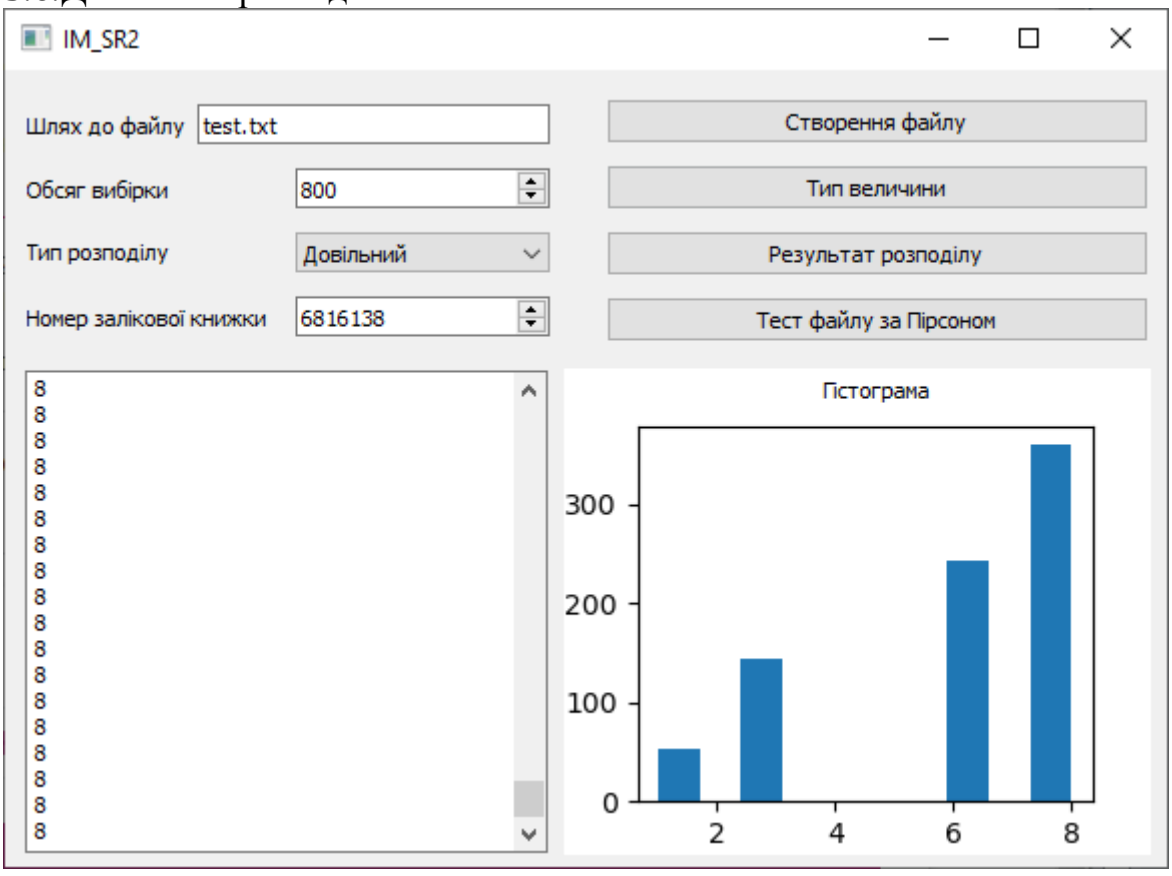
5.4.Рівномірний розподіл



5.5.Дискретний розподіл

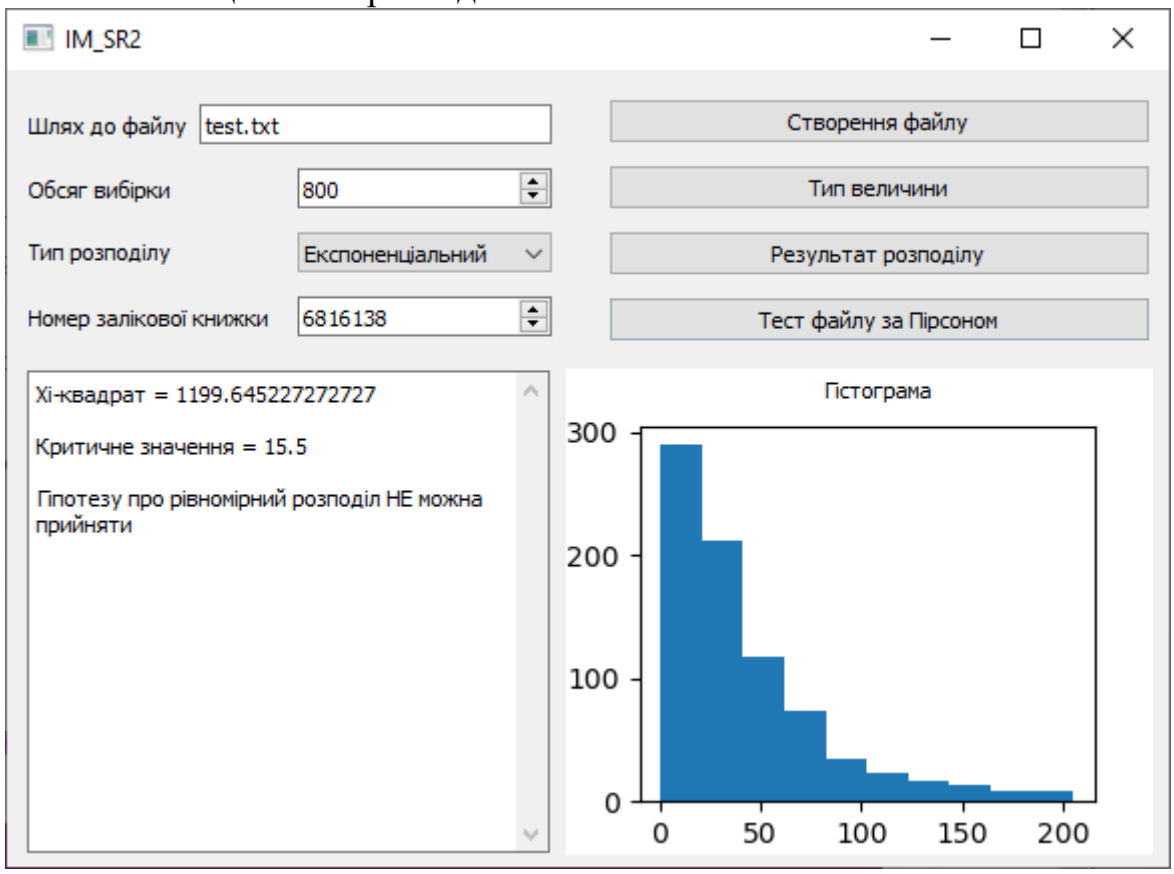


5.6.Довільний розподіл

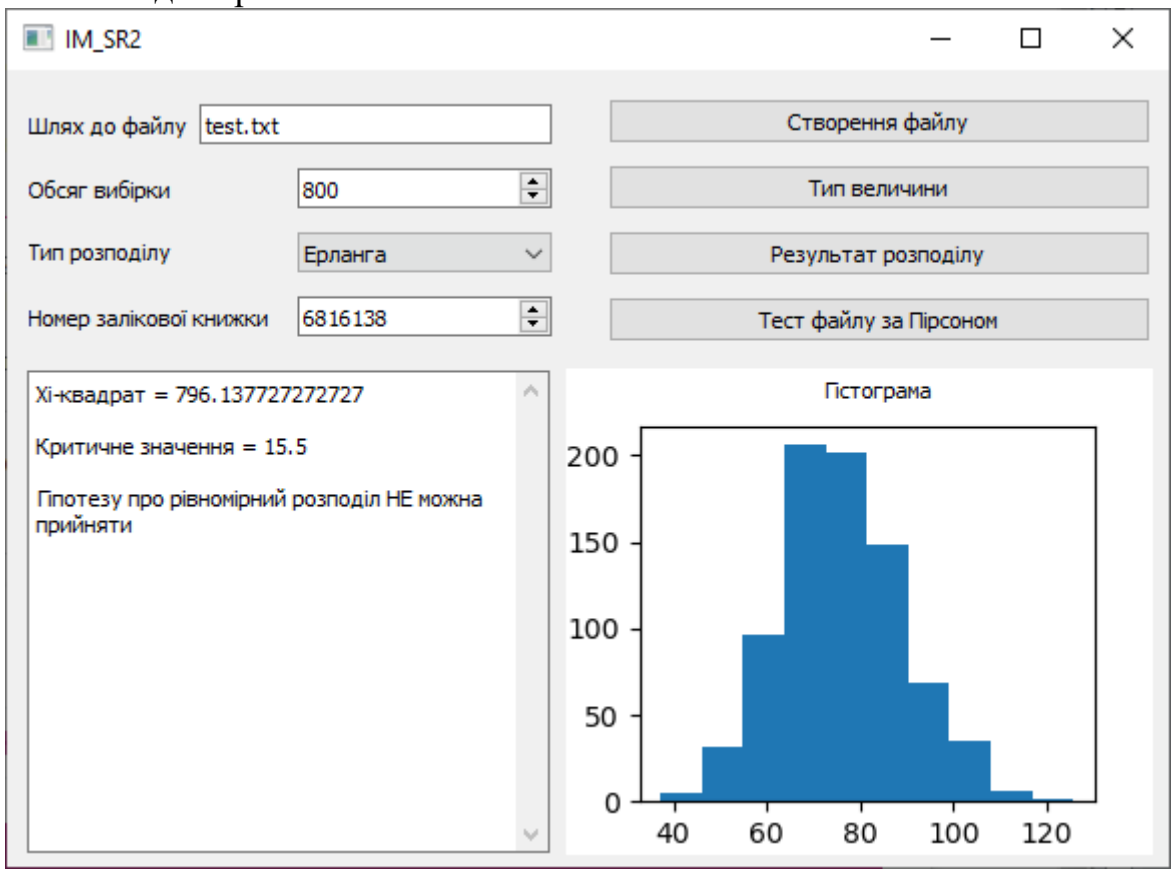


6. Тест за критерієм Пірсона

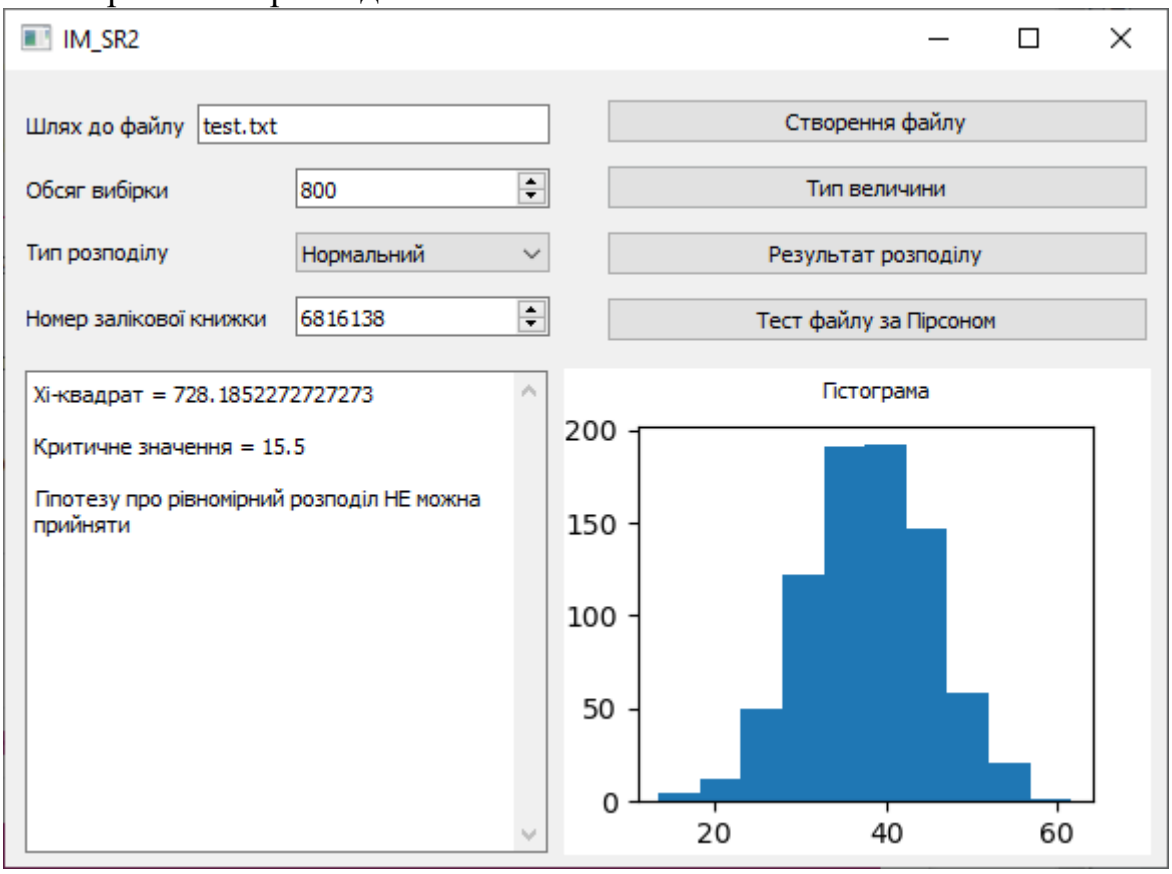
6.1.Експоненціальний розподіл



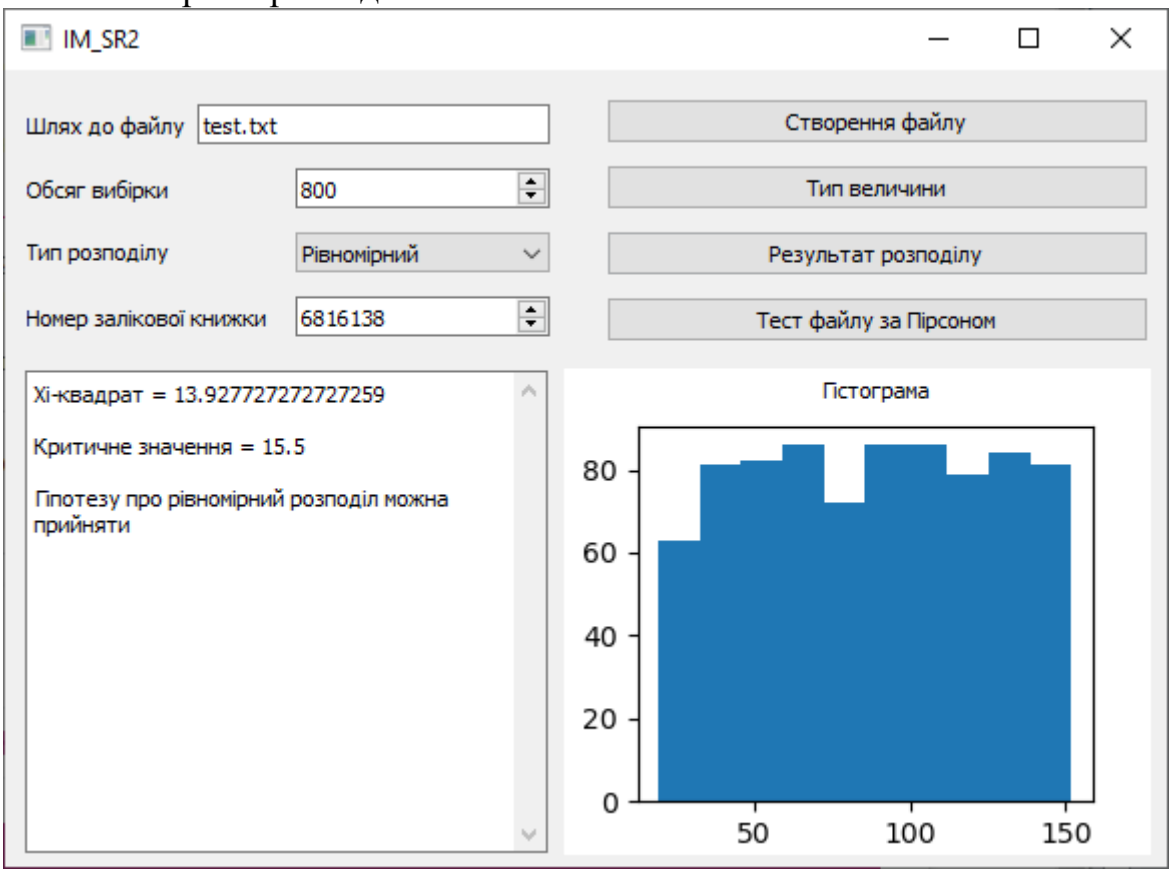
6.2.Розподіл Ерланга



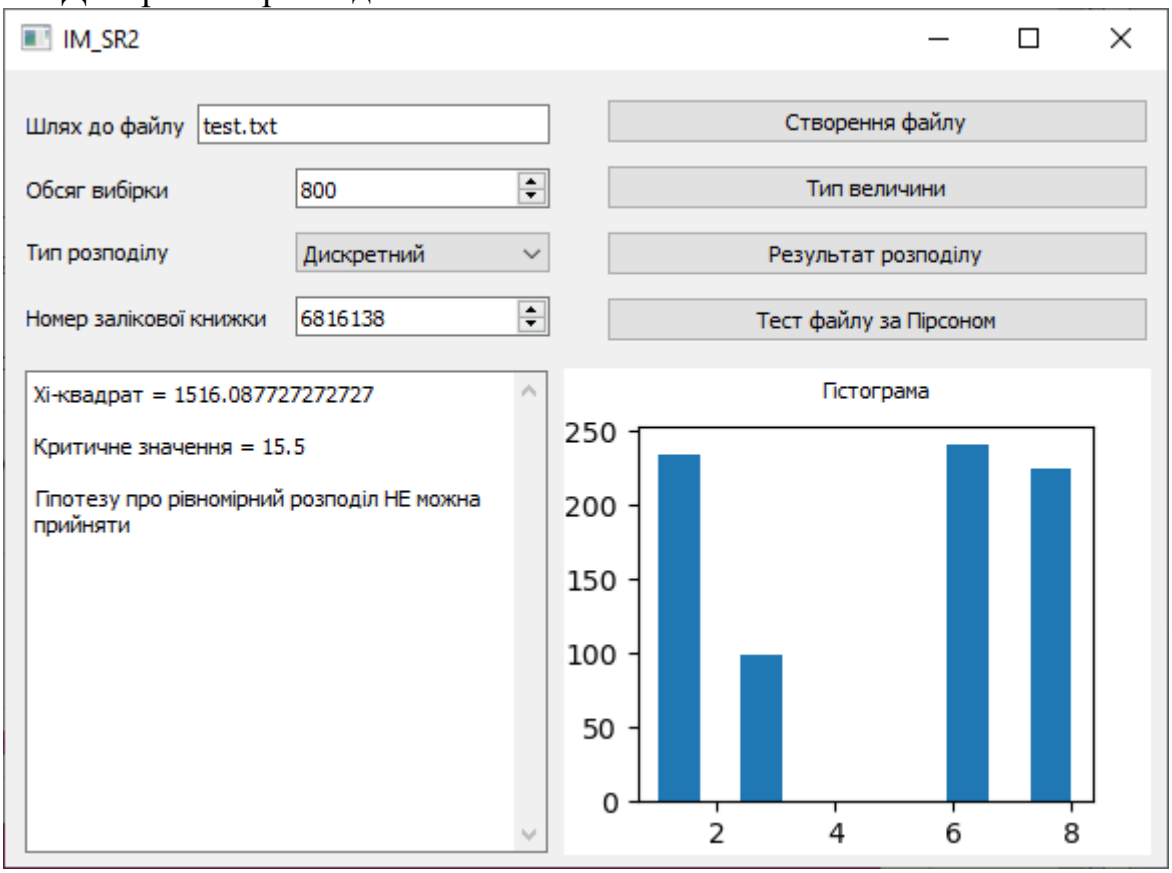
6.3.Нормальний розподіл



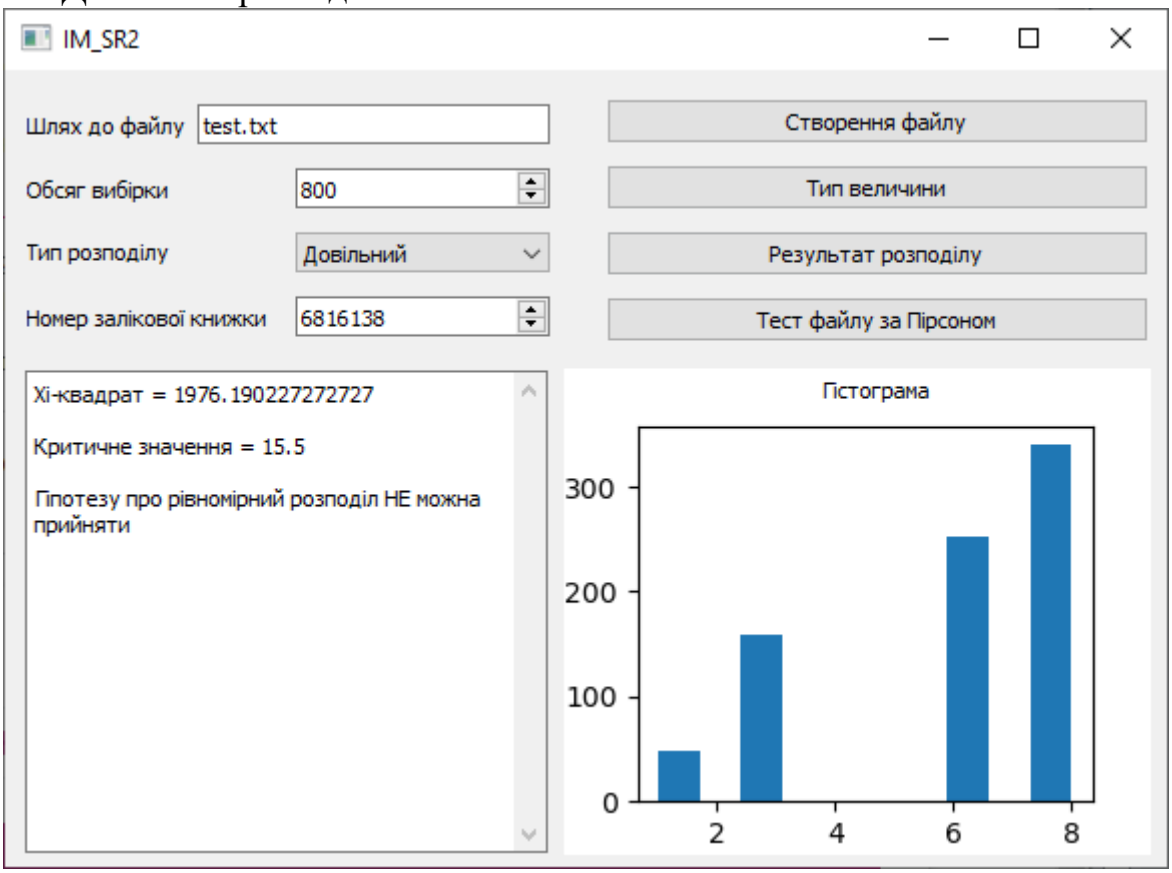
6.4.Рівномірний розподіл



6.5.Дискретний розподіл



6.6.Довільний розподіл



7. Код програми

```
from random import *
import sys
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import *
from IM_SR2_ui import Ui_MainWindow
import numpy as np
import json

def digits_recursive(n, digits=None):
    if digits is None:
        digits = []
    return digits_recursive(n // 10, [n % 10] + digits) if n else digits or [0]

def discrete_variate(Seed):
    discrete_values = digits_recursive(Seed)
    basic_p = 1/len(discrete_values)
    p_dict = {}
    for i in discrete_values:
        if i in p_dict:
            p_dict[i] += 1
        else:
            p_dict[i] = 1
    prev_p = 0
    for key in p_dict:
        prev_p += p_dict[key]*basic_p
        p_dict[key] = prev_p
    rv = random()
    interval_i = 0
    for key in p_dict:
        if rv <= p_dict[key]:
            rv = key
            break
        else:
            interval_i += 1
    return rv

def arbitrary_distribution(Seed):
    discrete_values = sorted(list(set(digits_recursive(Seed))))
    basic_p = 1 / (2*len(discrete_values))
    p_dict = {}
    for i in discrete_values:
        if i in p_dict:
            p_dict[i] += 1
        else:
            p_dict[i] = 1
    prev_p = 0
    shift = 0
    for key in p_dict:
        prev_p += p_dict[key] * basic_p + shift*2*basic_p
        p_dict[key] = prev_p
        shift += 1
    rv = uniform(0, 2)
    interval_i = 0
    for key in p_dict:
        if rv <= p_dict[key]:
            rv = key
            break
        else:
            interval_i += 1
    return rv
```

```
class MyWindow(QMainWindow, Ui_MainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        self.setupUi(self)
        self.setWindowTitle("IM_SR2")
        self.pushButton.clicked.connect(self.create_file)
        self.pushButton_2.clicked.connect(self.view_file)
        self.pushButton_3.clicked.connect(self.view_params_file)
        self.pushButton_4.clicked.connect(self.pirson_test)

    def create_file(self):
        N = self.spinBox.value()
        Zach = self.spinBox_2.value()
        Last_two = Zach % 100
        betas = {
            0: 2,
            1: 3
        }
        beta = betas[Last_two % 2]
        variates = {
            0: [expovariate(1/Last_two) for _ in range(N)],
            1: [gammavariate(Last_two, beta) for _ in range(N)],
            2: [normalvariate(Last_two, Last_two*0.2) for _ in range(N)],
            3: [uniform(Last_two/2, Last_two*4) for _ in range(N)],
            4: [discrete_variate(Zach) for _ in range(N)],
            5: [arbitrary_distribution(Zach) for _ in range(N)],
        }
        X = variates[self.comboBox.currentIndex()]
        X = sorted(X)
        self.mplWidget.canvas.axes.clear()
        self.mplWidget.canvas.axes.hist(X)
        self.mplWidget.canvas.draw()

        filename = self.lineEdit.text()
        with open(filename, 'w') as fw:
            json.dump(X, fw)

    def view_file(self):
        filename = self.lineEdit.text()
        with open(filename, 'r') as fr:
            X = json.load(fr)
        if len(X) == len(set(X)):
            self.textBrowser.setText("Неперервна величина")
        else:
            self.textBrowser.setText("Дискретна величина")

    def view_params_file(self):
        filename = self.lineEdit.text()
        with open(filename, 'r') as fr:
            X = json.load(fr)
        self.textBrowser.setText("")
        for x in X:
            self.textBrowser.append(str(x))
        self.mplWidget.canvas.axes.clear()
        self.mplWidget.canvas.axes.hist(X)
        self.mplWidget.canvas.draw()

    def pirson_test(self):
        filename = self.lineEdit.text()
        with open(filename, 'r') as fr:
```

```
            X = json.load(fr)

        freq, bins = np.histogram(X)
        theoretical_frequency = len(X)/len(bins)
        critical_distribution_points = {1: 0.05, 2: 3.8, 3: 6.0, 4: 7.8, 5: 9.5, 6:
11.1, 7: 12.6, 8: 14.1,
                                        9: 15.5, 10: 16.9, 11: 19.7, 12: 21.0, 13:
22.4, 14: 23.7, 15: 25.0,
                                        16: 26.3, 17: 27.6, 18: 28.9, 19: 30.1, 20:
31.4, 21: 32.7, 22: 33.9,
                                        23: 35.2, 24: 36.4, 25: 37.7, 26: 38.9, 27:
40.1, 28: 41.3, 29: 42.6,
                                        30: 43.8}
        xi2 = 0
        for i in range(len(bins) - 1):
            xi2 += (freq[i] - theoretical_frequency) ** 2 / theoretical_frequency
        self.textBrowser.setText("Xi-квадрат = " + str(xi2))
        degree_of_freedom = (len(bins) - 1) - 1
        critical_point = critical_distribution_points.get(degree_of_freedom)
        self.textBrowser.append("\nКритичне значення = " + str(critical_point))

        if xi2 < critical_point:
            self.textBrowser.append("\nГіпотезу про рівномірний розподіл можна
прийняти")
        else:
            self.textBrowser.append("\nГіпотезу про рівномірний розподіл НЕ можна
прийняти")


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    my_app = MyWindow()
    my_app.show()
    sys.exit(app.exec_())
```

Висновок: в ході виконання самостійної роботи ми ознайомились із алгоритмами роботи генераторів випадкових чисел, що мають різні закони розподілення та методикою їх тестування. Ми виконали тест за критерієм Пірсона для випадкових величин згенерованих за різними законами розподілу і для всіх законів розподілу окрім рівномірного Хі-квадрат був значно більше критичного значення.