Московский авиационный институт

(национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 804 «Теория вероятностей и компьютерное моделирование»

Курсовая работа на тему

«Метод наименьших квадратов»

Студент: Сайфуллин И.К.

Группа: М8О-303Б-21

Дата: 20.12.2023

Оценка: \_\_\_\_\_

Москва, 2023

Оглавление

[НОРМАЛЬНОЕ РАСПРЕДЕЛЕНИЕ 2](#_Toc153965889)

[Номер 1 2](#_Toc153965890)

[Номер 2 6](#_Toc153965891)

[Номер 3 8](#_Toc153965892)

[Номер 4 10](#_Toc153965893)

[Номер 5 10](#_Toc153965894)

[Номер 6 11](#_Toc153965895)

[Номер 7 11](#_Toc153965896)

[Листинг программы: 12](#_Toc153965897)

# **НОРМАЛЬНОЕ РАСПРЕДЕЛЕНИЕ**

Модель полезного сигнала:

Модель наблюдений:

# **Номер 1**

Найдем старший порядок многочлена , используя критерий Фишера, и вычислим оценки известных параметров , используя метод наименьших квадратов.

Для подбора старшей степени многочлена используем критерий Фишера. Он имеет вид:

Статистика критерия имеет вид:

где - объем выборки, – выборка, – матрица МНК-оценок параметров , – элемент главной диагонали .

Заполняем :

Найдем матрицу

Будем считать точечные МНК-оценки для каждого порядка по очереди, начиная с 1.

p = 1

Уровень значимости = 0.05, значит, уровень надежности 0.95

Статистика попала в критическую область. Идем дальше.

p = 2

Уровень значимости = 0.05, значит, уровень надежности 0.95

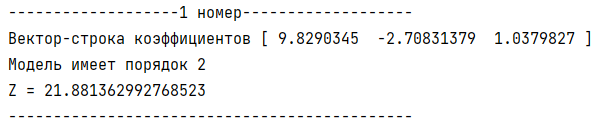
Статистика попала в критическую область. Идем дальше.

p = 3

Уровень значимости = 0.05, значит, уровень надежности 0.95

Статистика не попала в критическую область. Порядок: 2

Работа программы:



# 

# Номер 2

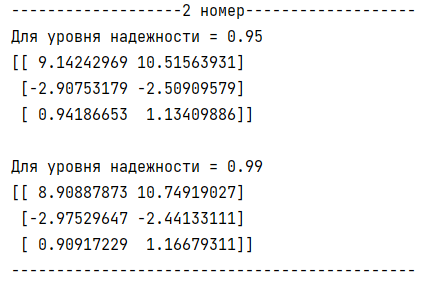
Используем следующую формулу для построения доверительных интервалов параметров в предположении нормальности ошибок.

где – k-й элемент главной диагонали матрицы .

Для уровня надежности = 0.95:

Для уровня надежности = 0.99:

Работа программы:



# Номер 3

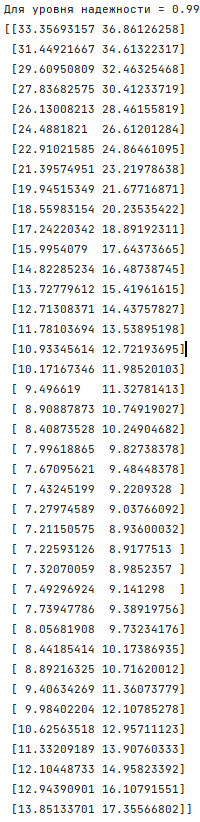
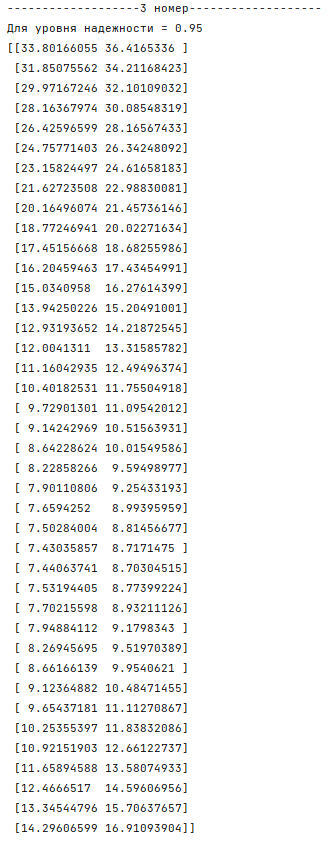
Используем следующую формулу для построения доверительных интервалов полезного сигнала в предположении нормальности ошибок.

где

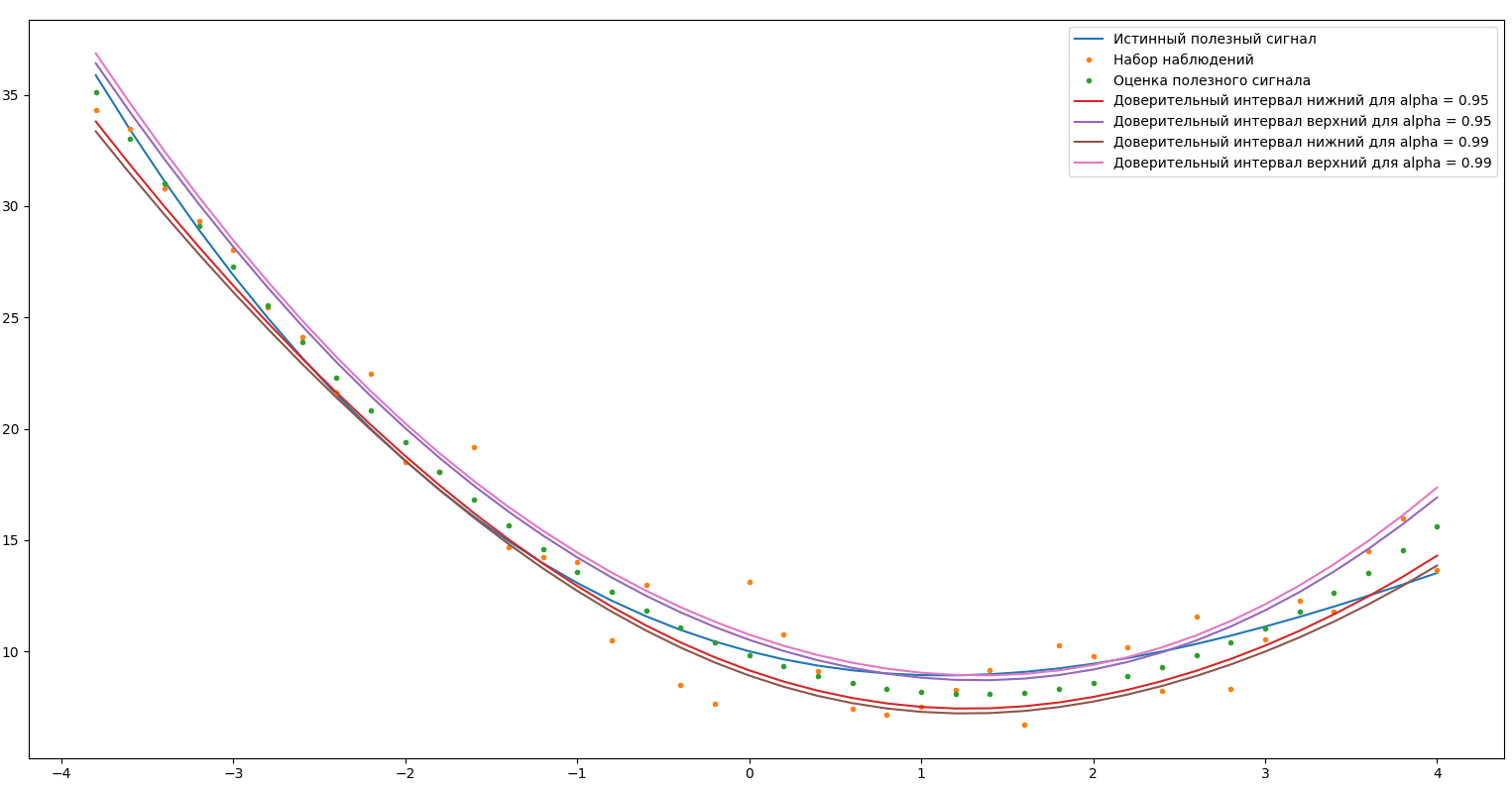
Для уровня надежности = 0.95:

Для уровня надежности = 0.99:

Работа программы:

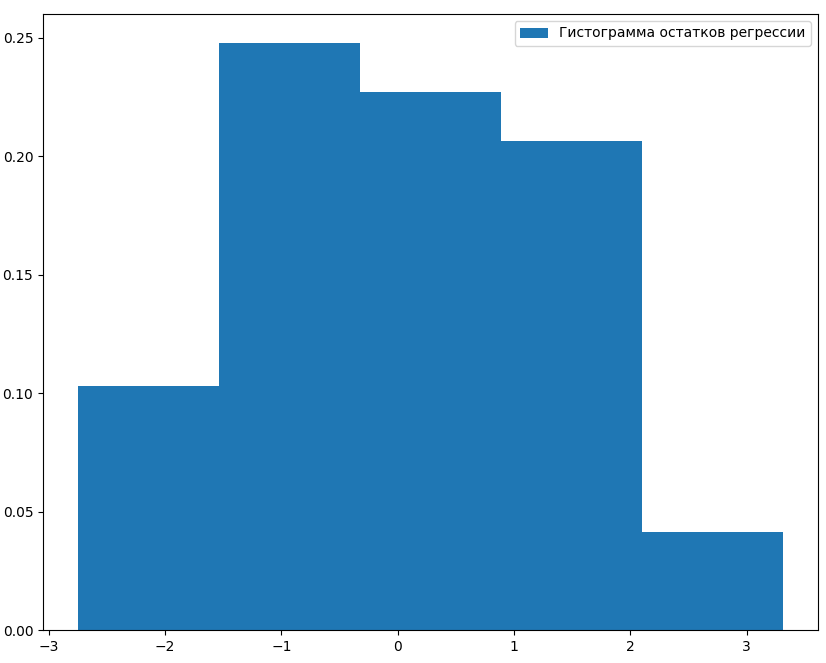


# Номер 4

****

# Номер 5

Остатки регрессии – это разность между наблюдаемыми значения и значениями, предсказанными изучаемой регрессионной моделью.

****

# Номер 6

Норма E:

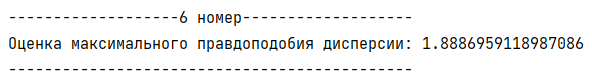
В предположении нормальности мы можем вычислить оценку максимального правдоподобия случайной ошибки следующим образом:

Для логарифмическая функция правдоподобия имеет вид:

C учетом того, что найдем экстремум (без проверки достаточных условий)

откуда из первого уравнения выражается МП-оценка и уже из второго:

Работа программы:



# Номер 7

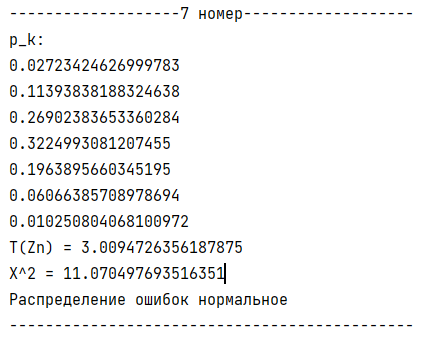
Для проверки нормальности распределения ошибок используем - критерий Пирсона.

Статистика считается по формуле:

где

Статистика попала в доверительный интервал. Следовательно, закон распределения ошибок нормальный.

Работа программы:



# Листинг программы:

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.stats import t, norm, chi2, f  
  
EPS = np.array(  
 [-1.57570224093566,  
 0.026817652058368,  
 -0.327083400768208,  
 0.384457963489581,  
 1.13092869940426,  
 0.496851264054229,  
 0.907339535512404,  
 0.100098748083528,  
 2.49180275180069,  
 -0.0728842702874212,  
 0.826877669546407,  
 3.12941087626926,  
 -0.290112040544173,  
 0.273222919888945,  
 0.945511583398219,  
 -1.76916070866538,  
 1.40708789619999,  
 -2.48341591842828,  
 -2.77630227137947,  
 3.13991491474395,  
 1.10107674176732,  
 -0.257722179163009,  
 -1.73620920255182,  
 -1.84146124965671,  
 -1.43220867624349,  
 -0.657551650718961,  
 0.186855475002093,  
 -2.35604061658699,  
 1.03240424046648,  
 0.334456144504168,  
 0.470946667999632,  
 -1.76644417119997,  
 1.23171894652158,  
 -2.41344098737007,  
 -0.553993401124847,  
 0.712286159261229,  
 -0.229382238056625,  
 2.02815685186965,  
 2.95489535309773,  
 0.147225408140916  
 ])  
  
n = 40  
sigma = 1.6 \*\* 0.5  
theta0 = 10  
theta1 = -2  
theta2 = 1  
theta3 = -0.07  
  
theta = np.array([theta0, theta1, theta2, theta3])  
  
  
def y(th0, th1, th2, th3, t):  
 return th3 \* t \*\* 3 + th2 \* t \*\* 2 + th1 \* t + th0  
  
  
def task\_1():  
 X = np.zeros((n, 6))  
  
 for i in range(n):  
 X[i][0] = 1  
 X[i][1] = (-4 + (i + 1) \* 8 / n)  
 X[i][2] = (-4 + (i + 1) \* 8 / n) \*\* 2  
 X[i][3] = (-4 + (i + 1) \* 8 / n) \*\* 3  
 X[i][4] = (-4 + (i + 1) \* 8 / n) \*\* 4  
 X[i][5] = (-4 + (i + 1) \* 8 / n) \*\* 5  
  
 Y = np.dot(X[:n, :4], theta) + EPS  
  
 def get\_stat(p, alpha=0.05):  
 X\_m = X[:n, :p + 1]  
  
 Theta\_m = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(X\_m), X\_m)), np.transpose(X\_m)), Y)  
  
 quant\_m = f.ppf(1 - alpha, 1, n - (p + 1))  
  
 alpha\_m = np.linalg.inv(np.dot(np.transpose(X\_m), X\_m))[p, p]  
  
 hatE = Y - np.dot(X\_m, Theta\_m)  
  
 NormaE = np.dot(np.transpose(hatE), hatE)  
 CentralStatistics = Theta\_m[p] / (alpha\_m \* NormaE) \*\* 0.5 \* (n - (p + 1)) \*\* 0.5  
 *# print(p, Theta\_m, CentralStatistics, quant\_m)* return Theta\_m, CentralStatistics, quant\_m  
  
 p = 1  
 while True:  
 Theta, Z, quant = get\_stat(p)  
 if abs(Z) > quant:  
 p += 1  
 else:  
 return \*get\_stat(p - 1)[:2], p - 1  
  
  
def task\_2(p):  
 X = np.zeros((40, 4))  
  
 for i in range(40):  
 X[i][0] = 1  
 X[i][1] = (-4 + (i + 1) \* 8 / n)  
 X[i][2] = (-4 + (i + 1) \* 8 / n) \*\* 2  
 X[i][3] = (-4 + (i + 1) \* 8 / n) \*\* 3  
  
 Y = np.dot(X, theta) + EPS  
 X\_m = X[:40, :p + 1]  
 Theta\_m = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(X\_m), X\_m)), np.transpose(X\_m)), Y)  
  
 hatE = Y - np.dot(X\_m, Theta\_m)  
  
 NormaE = np.dot(np.transpose(hatE), hatE)  
  
 def get\_interval(p, level):  
 interval = []  
 quant\_m = t.ppf(  
 q=1 - (1 - level) / 2,  
 df=n - (p + 1))  
 for i in range(p + 1):  
 left = Theta\_m[i] - quant\_m \* (NormaE \* np.linalg.inv(np.dot(np.transpose(X\_m), X\_m))[i, i]) \*\* 0.5 / (  
 n - (p + 1)) \*\* 0.5  
 right = Theta\_m[i] + quant\_m \* (NormaE \* np.linalg.inv(np.dot(np.transpose(X\_m), X\_m))[i, i]) \*\* 0.5 / (  
 n - (p + 1)) \*\* 0.5  
  
 interval.append([left, right])  
  
 return np.array(interval)  
  
 return get\_interval(p, level=0.95), get\_interval(p, level=0.99)  
  
  
def task\_3(p, hatTheta):  
 X = np.zeros((40, 4))  
  
 for i in range(40):  
 X[i][0] = 1  
 X[i][1] = (-4 + (i + 1) \* 8 / n)  
 X[i][2] = (-4 + (i + 1) \* 8 / n) \*\* 2  
 X[i][3] = (-4 + (i + 1) \* 8 / n) \*\* 3  
  
 Y = np.dot(X, theta) + EPS  
 X\_m = X[:40, :p + 1]  
  
 hatY = np.dot(X\_m, hatTheta)  
  
 Theta\_m = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(X\_m), X\_m)), np.transpose(X\_m)), Y)  
  
 def alpha(i):  
 return np.dot(np.dot(X\_m[i], np.linalg.inv(np.dot(np.transpose(X\_m), X\_m))), np.transpose(X\_m[i]))  
  
 hatE = Y - np.dot(X\_m, Theta\_m)  
  
 NormaE = np.dot(np.transpose(hatE), hatE)  
  
 def get\_interval(p, level):  
 left = np.zeros(n)  
 right = np.zeros(n)  
 quant\_m = t.ppf(q=1 - (1 - level) / 2, df=n - (p + 1))  
 for i in range(n):  
 left[i] = hatY[i] - quant\_m \* (NormaE \* alpha(i)) \*\* 0.5 / (n - (p + 1)) \*\* 0.5  
 right[i] = hatY[i] + quant\_m \* (NormaE \* alpha(i)) \*\* 0.5 / (n - (p + 1)) \*\* 0.5  
  
 return np.array([left, right]).transpose()  
  
 return get\_interval(p, level=0.95), get\_interval(p, level=0.99)  
  
  
def task\_4(theta\_m, level\_095, level\_099):  
 X = np.zeros((40, 4))  
  
 for i in range(40):  
 X[i][0] = 1  
 X[i][1] = (-4 + (i + 1) \* 8 / n)  
 X[i][2] = (-4 + (i + 1) \* 8 / n) \*\* 2  
 X[i][3] = (-4 + (i + 1) \* 8 / n) \*\* 3  
  
 X\_m = X[:40, :len(theta\_m)]  
  
 Y\_without\_noise = np.dot(X, theta)  
 Y\_with\_noise = np.dot(X, theta) + EPS  
  
 fig = plt.figure(figsize=[14, 10])  
 ax = fig.add\_subplot(1, 1, 1)  
  
 ax.plot(x := X.transpose()[1], Y\_without\_noise, label="Истинный полезный сигнал")  
 ax.plot(x, Y\_with\_noise, label="Набор наблюдений", ls='None', marker='.')  
 y\_check = np.dot(X\_m, theta\_m)  
 print(y\_check)  
 ax.plot(x, y\_check, label="Оценка полезного сигнала", ls='None', marker='.')  
 ax.plot(x, [level\_095[i][0] for i in range(n)], label="Доверительный интервал нижний для alpha = 0.95")  
 ax.plot(x, [level\_095[i][1] for i in range(n)], label="Доверительный интервал верхний для alpha = 0.95")  
 ax.plot(x, [level\_099[i][0] for i in range(n)], label="Доверительный интервал нижний для alpha = 0.99")  
 ax.plot(x, [level\_099[i][1] for i in range(n)], label="Доверительный интервал верхний для alpha = 0.99")  
  
 ax.legend()  
  
  
def task\_5():  
 X = np.zeros((40, 4))  
  
 for i in range(40):  
 X[i][0] = 1  
 X[i][1] = (-4 + (i + 1) \* 8 / n)  
 X[i][2] = (-4 + (i + 1) \* 8 / n) \*\* 2  
 X[i][3] = (-4 + (i + 1) \* 8 / n) \*\* 3  
  
 Y = np.dot(X, theta) + EPS  
 X\_m = X[:40, :p + 1]  
 Theta\_m = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(X\_m), X\_m)), np.transpose(X\_m)), Y)  
 fig = plt.figure(figsize=[10, 8])  
 ax = fig.add\_subplot(1, 1, 1)  
 *# Остатки регрессии* hatE = Y - np.dot(X\_m, Theta\_m)  
 a = ax.hist(hatE, bins=5, density=True, label="Гистограмма остатков регрессии")  
 ax.legend()  
 return a  
  
  
def task\_6(p):  
 X = np.zeros((40, 4))  
  
 for i in range(40):  
 X[i][0] = 1  
 X[i][1] = (-4 + (i + 1) \* 8 / n)  
 X[i][2] = (-4 + (i + 1) \* 8 / n) \*\* 2  
 X[i][3] = (-4 + (i + 1) \* 8 / n) \*\* 3  
  
 Y = np.dot(X, theta) + EPS  
 X\_m = X[:40, :p + 1]  
 Theta\_m = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(X\_m), X\_m)), np.transpose(X\_m)), Y)  
  
 hatE = Y - np.dot(X\_m, Theta\_m)  
  
 NormaE = np.dot(np.transpose(hatE), hatE)  
  
 *# print(NormaE)  
 # print(hatE, np.var(hatE))  
 # dispersion = np.var(hatE)* dispersion = NormaE/n  
  
 return dispersion  
  
  
def task\_7(gistogramma):  
 *# print(gistogramma)* X = np.zeros((40, 4))  
  
 for i in range(40):  
 X[i][0] = 1  
 X[i][1] = (-4 + (i + 1) \* 8 / n)  
 X[i][2] = (-4 + (i + 1) \* 8 / n) \*\* 2  
 X[i][3] = (-4 + (i + 1) \* 8 / n) \*\* 3  
  
 Y = np.dot(X, theta) + EPS  
 X\_m = X[:40, :p + 1]  
 Theta\_m = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(X\_m), X\_m)), np.transpose(X\_m)), Y)  
  
 hatE = Y - np.dot(X\_m, Theta\_m)  
  
 NormaE = np.dot(np.transpose(hatE), hatE)  
  
 sigma = NormaE / n  
 *# print(sigma)* print("p\_k:")  
 def T(gistogramma):  
 T = 0  
 for i in range(-1, len(gistogramma[0]) + 1):  
 if i == -1:  
 T += (pk := (norm.cdf(gistogramma[1][i + 1] / sigma \*\* 0.5) - norm.cdf(-100 / sigma \*\* 0.5))) \*\* 2 / pk  
  
 elif i == len(gistogramma[0]):  
 T += (pk := (norm.cdf(+ 100 / sigma \*\* 0.5) - norm.cdf(gistogramma[1][i] / sigma \*\* 0.5))) \*\* 2 / pk  
  
 else:  
 T += (pk := (norm.cdf(gistogramma[1][i + 1] / sigma \*\* 0.5) - norm.cdf(gistogramma[1][i] / sigma \*\* 0.5)) - gistogramma[0][i]) \*\* 2 / pk  
 print(pk)  
 return T \* n  
 TZn = T(gistogramma)  
 *# print(TZn)  
 # print(chi2.ppf(0.95, 5))* return "Распределение ошибок нормальное" if 0 < TZn < chi2.ppf(0.95,  
 5) else "Распределение ошибок не является нормальным"  
  
  
print("-------------------1 номер-------------------")  
Theta, Z, p = task\_1()  
print(f"Вектор-строка коэффициентов {Theta}\nМодель имеет порядок {p}\nZ = {Z}")  
print("---------------------------------------------")  
print("-------------------2 номер-------------------")  
level\_095, level\_099 = task\_2(p)  
print(f"Для уровня надежности = 0.95\n{level\_095}\n")  
print(f"Для уровня надежности = 0.99\n{level\_099}")  
print("---------------------------------------------")  
print("-------------------3 номер-------------------")  
level\_095\_for\_signal, level\_099\_for\_signal = task\_3(p, Theta)  
print(f"Для уровня надежности = 0.95\n{level\_095\_for\_signal}\n")  
print(f"Для уровня надежности = 0.99\n{level\_099\_for\_signal}")  
print("---------------------------------------------")  
print("-------------------4 номер-------------------")  
task\_4(Theta, level\_095\_for\_signal, level\_099\_for\_signal)  
print("---------------------------------------------")  
print("-------------------5 номер-------------------")  
gistogramma = task\_5()  
print("---------------------------------------------")  
print("-------------------6 номер-------------------")  
dispersion = task\_6(p)  
print(f"Оценка максимального правдоподобия дисперсии: {dispersion}")  
print("---------------------------------------------")  
print("-------------------7 номер-------------------")  
print(task\_7(gistogramma))  
print("---------------------------------------------")  
plt.show()