Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины «Алгоритмизация»
Вариант 29

Выполнил:
Саенко Андрей Максимович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», заочная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А., канд. технических
наук, доцент кафедры
инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Метод наименьших квадратов

Цель: Научиться применять метод наименьших квадратов и находить коэффициент парной корреляции.

Порядок выполнения работы:

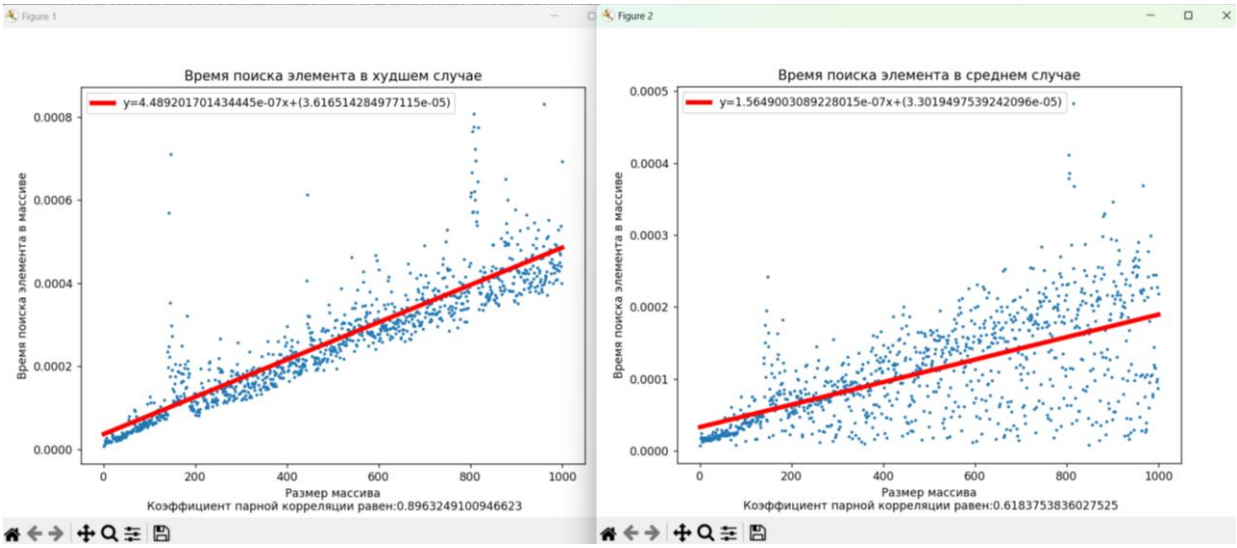

Рисунок 1 – Результат работы программы

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import random
import numpy as np
import timeit
import matplotlib.pyplot as plt
from math import sqrt
def correlation(array_of_values_x,array_of_values_y):
    sigma_first = 0
    sigma_second = 0
    sigma_third = 0
    average_value_x = 0
    average_value_y = 0
    sum_x = 0
    sum_y = 0

    for i in range(len(array_of_values_x)):
        sum_x += array_of_values_x[i]
        sum_y += array_of_values_y[i]
    average_value_x = sum_x/len(array_of_values_x)
    average_value_y = sum_y/len(array_of_values_y)

    for i in range(len(array_of_values_x)):
        sigma_first += ((array_of_values_x[i] - average_value_x)*
                        (array_of_values_y[i] - average_value_y))
        sigma_second += (array_of_values_x[i]-average_value_x)**2
        sigma_third += (array_of_values_y[i]-average_value_y)**2

    pair_correlation_coefficient = (sigma_first/(sqrt(sigma_second)*
                                                 sqrt(sigma_third)))
    return pair_correlation_coefficient
def linear_search(array,element):
    for i in range(0,len(array)):
        if array[i] == element:
```

```
            return 1
    return 0
array_of_search_time_bad = []
amount_of_elements = []
array_of_search_time_avr = []
for i in range(1,1001):
    arr=[0 for i in range(0,i)]
    amount_of_elements.append(i)
    for j in range(0,len(arr)):
        arr[j] = random.randint(500,1000)
    linear_search(arr,1)
    search_time = (timeit.timeit(lambda: linear_search(arr,1), number=50))/50
    print("Время поиска в массиве из ",i,
          " элементов в худшем случае: ", search_time)
    array_of_search_time_bad.append(search_time)
    search_time=(timeit.timeit(lambda: linear_search(
        arr,arr[round(len(arr)/2)]), number=50))/50
    print("Время поиска в массиве из ",i,
          " элементов в среднем случае: ",search_time,"\n")
    array_of_search_time_avr.append(search_time)
sum_search_time_bad=sum(array_of_search_time_bad)
sum_amt_of_el=sum(amount_of_elements)
sum_of_sqr_amt_of_el=0
sum_mltp_amt_time=0
bn=len(amount_of_elements)

for i in amount_of_elements:
    sum_of_sqr_amt_of_el+=i*i

for i in range(0,len(array_of_search_time_bad)):
    sum_mltp_amt_time+=amount_of_elements[i]*array_of_search_time_bad[i]
matrix_bad = np.array([[sum_of_sqr_amt_of_el, sum_amt_of_el],
                       [sum_amt_of_el, bn]])
det_bad = np.linalg.det(matrix_bad)
first_mt_bad = np.array([[sum_mltp_amt_time, sum_amt_of_el],
                         [sum_search_time_bad, bn]])
first_det_bad=np.linalg.det(first_mt_bad)
second_mt_bad = np.array([[sum_of_sqr_amt_of_el, sum_mltp_amt_time],
                          [sum_amt_of_el, sum_search_time_bad]])
second_det_bad=np.linalg.det(second_mt_bad)
first_coefficient_bad=first_det_bad/det_bad
second_coefficient_bad=second_det_bad/det_bad
func_bad=[]
for i in range(1,1001):
    func_bad.append(first_coefficient_bad*(i)+second_coefficient_bad)
sum_search_time_avr = sum(array_of_search_time_avr)
sum_mltp_amt_time_avr = 0

for i in range(0,len(array_of_search_time_bad)):
    sum_mltp_amt_time_avr+=amount_of_elements[i]*array_of_search_time_avr[i]
matrix_avr = np.array([[sum_of_sqr_amt_of_el, sum_amt_of_el],
                       [sum_amt_of_el, bn]])
det_avr = np.linalg.det(matrix_avr)
first_mt_avr = np.array([[
    sum_mltp_amt_time_avr, sum_amt_of_el],
   [sum_search_time_avr, bn]])
first_det_avr = np.linalg.det(first_mt_avr)
second_mt_avr = np.array([[sum_of_sqr_amt_of_el, sum_mltp_amt_time_avr],
                          [sum_amt_of_el, sum_search_time_avr]])
second_det_avr = np.linalg.det(second_mt_avr)
first_coefficient_avr = first_det_avr/det_avr
second_coefficient_avr = second_det_avr/det_avr
func_average=[]
for i in range(1,1001):
    func_average.append(first_coefficient_avr*i +
                        second_coefficient_avr)
plt.figure(figsize=(8,6))
```

```
plt.figure(1)
plt.title("Время поиска элемента в худшем случае")
plt.plot(amount_of_elements,func_bad,color='red',linewidth=4)
plt.scatter(amount_of_elements, array_of_search_time_bad,s=3)
plt.xlabel('Размер массива\n Коэффициент парной корреляции равен:'+
           str(correlation(amount_of_elements,array_of_search_time_bad)))
plt.legend(['y='+str(first_coefficient_bad)+"x+("+
            str(second_coefficient_bad)+")"])
plt.ylabel("Время поиска элемента в массиве")
plt.figure(figsize=(8,6))
plt.figure(2)
plt.title("Время поиска элемента в среднем случае")
plt.plot(amount_of_elements,func_average,color='red',linewidth=4)
plt.scatter(amount_of_elements,array_of_search_time_avr,s=3)
plt.xlabel('Размер массива\n Коэффициент парной корреляции равен:'+
           str(correlation(amount_of_elements,array_of_search_time_avr)))
plt.legend(['y='+str(first_coefficient_avr)+"x+("+
            str(second_coefficient_avr)+")"])
plt.ylabel("Время поиска элемента в массиве")
plt.show()
```

Вывод

Время поиска элемента в массиве увеличивается линейно в худшем и среднем случаях при увеличении размера массива, если используется алгоритм линейного поиска. Коэффициент парной корреляции позволяет определить связь между величинами, а метод наименьших квадратов позволяет построить график зависимости на основе экспериментальных данных.