Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Кафедра Систем Управления и Информатики

Лабораторная работа №2

Вариант №1

Выполнили:

Волгин Л.А.

Шляхов Д.О.

Проверил:

Мусаев А.А.

Санкт-Петербург,

2022

https://github.com/Puroblast/Programming_hoomework/tree/main/Dz_2

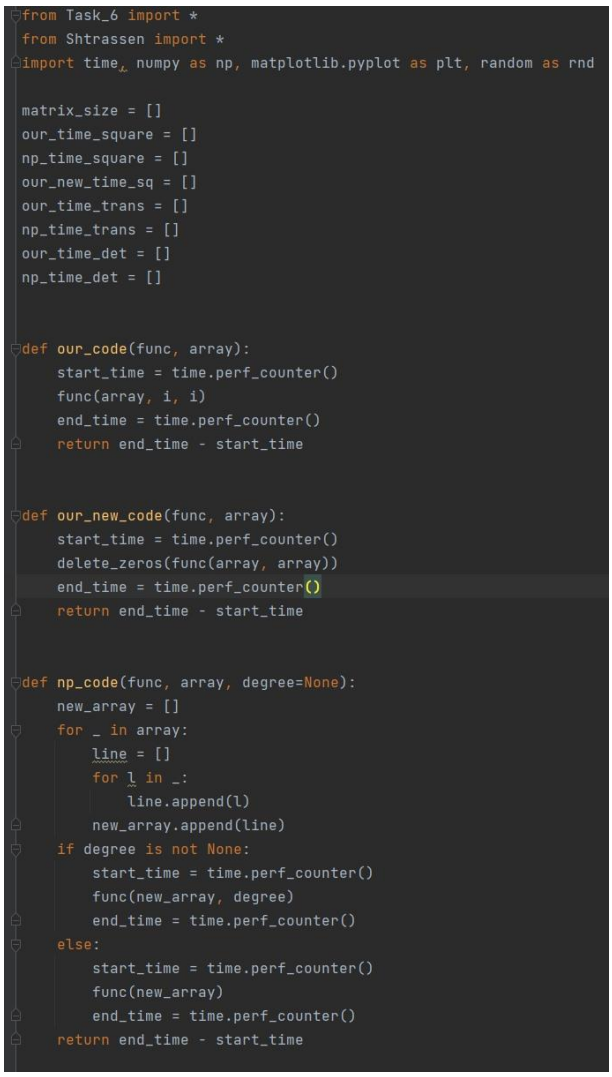
Задание 1

Описание задания:

Используя результаты из задания 6 лабораторной работы №1 создать модуль, позволяющий производить возведение в квадрат, транспонирование и нахождение определителя матрицы. Подключить данный модуль к новой программе. Проверить быстродействие реализованных функций. Реализовать аналогичные функции с помощью модуля NumPy. Сравнить быстродействие. Постараться улучшить результат. Исходная и улучшенная версия алгоритмов должна быть отражена на GitHub.

Листинг (Решение):

Для решения поставленной задачи был использован функционал библиотеки numpy, а так же модуль, представленный в виде программы для работы с матрицами, написанной нами. Были проведены сравнения скорости работ программ (Рисунок 1) и построены графики (Рисунок 2), по результатам которых мы пришли к выводу, что наша функция возведения в квадрат работает немного медленнее аналогичной функции из numpy. Нами было принято решение реализовать алгоритм Штрассена(Рисунки 3,4) – вычисление произведения двух матрицс помощью рекурсивной функции, в которой используются математические формулы для вычисления этого произведения. Сложность выполнения данного алгоритма – O(n^log2(7)). Этот алгоритм фактически должен работать быстрее.



```python
from Task_6 import *
from Shtrassen import *
import time, numpy as np, matplotlib.pyplot as plt, random as rnd

matrix_size = []
our_time_square = []
np_time_square = []
our_new_time_sq = []
our_time_trans = []
np_time_trans = []
our_time_det = []
np_time_det = []


def our_code(func, array):
    start_time = time.perf_counter()
    func(array, i, i)
    end_time = time.perf_counter()
    return end_time - start_time


def our_new_code(func, array):
    start_time = time.perf_counter()
    delete_zeros(func(array, array))
    end_time = time.perf_counter()
    return end_time - start_time


def np_code(func, array, degree=None):
    new_array = []
    for _ in array:
        line = []
        for l in _:
            line.append(l)
        new_array.append(line)
    if degree is not None:
        start_time = time.perf_counter()
        func(new_array, degree)
        end_time = time.perf_counter()
    else:
        start_time = time.perf_counter()
        func(new_array)
        end_time = time.perf_counter()
    return end_time - start_time
```

Рисунок 1. Основной код программы, сравнение скорости.

```
for i in range(2, 12):
    matrix = []
    matrix_size.append(i)
    for j in range(i):
        line = []
        for k in range(i):
            line.append(rnd.randint(1, 5))
        matrix.append(line)
    our_time_square.append(our_code(square, matrix))
    our_time_det.append(our_code(det, matrix))
    our_time_trans.append(our_code(transpon, matrix))
    our_new_time_sq.append(our_new_code(strassen, matrix))
    np_time_square.append(np_code(np.linalg.matrix_power, matrix, 2))
    np_time_det.append(np_code(np.linalg.det, matrix))
    np_time_trans.append(np_code(np.transpose, matrix))

fig, axs = plt.subplots(1, 3)
axs[0].set_title("Возведение в квадрат")
axs[0].set_xlabel("Размер матрицы")
axs[0].set_ylabel("Время выполнения")
axs[0].plot(matrix_size, np_time_square, label="Функция из NumPy")
axs[0].plot(matrix_size, our_time_square, label="Функция из Task_6")
axs[0].plot(matrix_size, our_new_time_sq, label="Штрассен")

axs[1].set_title("Нахождение определителя")
axs[1].set_xlabel("Размер матрицы")
axs[1].set_ylabel("Время выполнения")
axs[1].plot(matrix_size, np_time_det)
axs[1].plot(matrix_size, our_time_det)
axs[2].set_title("Транспонирование")
axs[2].set_xlabel("Размер матрицы")
axs[2].set_ylabel("Время выполнения")
axs[2].plot(matrix_size, np_time_trans)
axs[2].plot(matrix_size, our_time_trans)

fig.legend(fontsize=20,
           ncol=1,
           edgecolor='b',
           title='Прямые',
           title_fontsize='15')
plt.show()
```

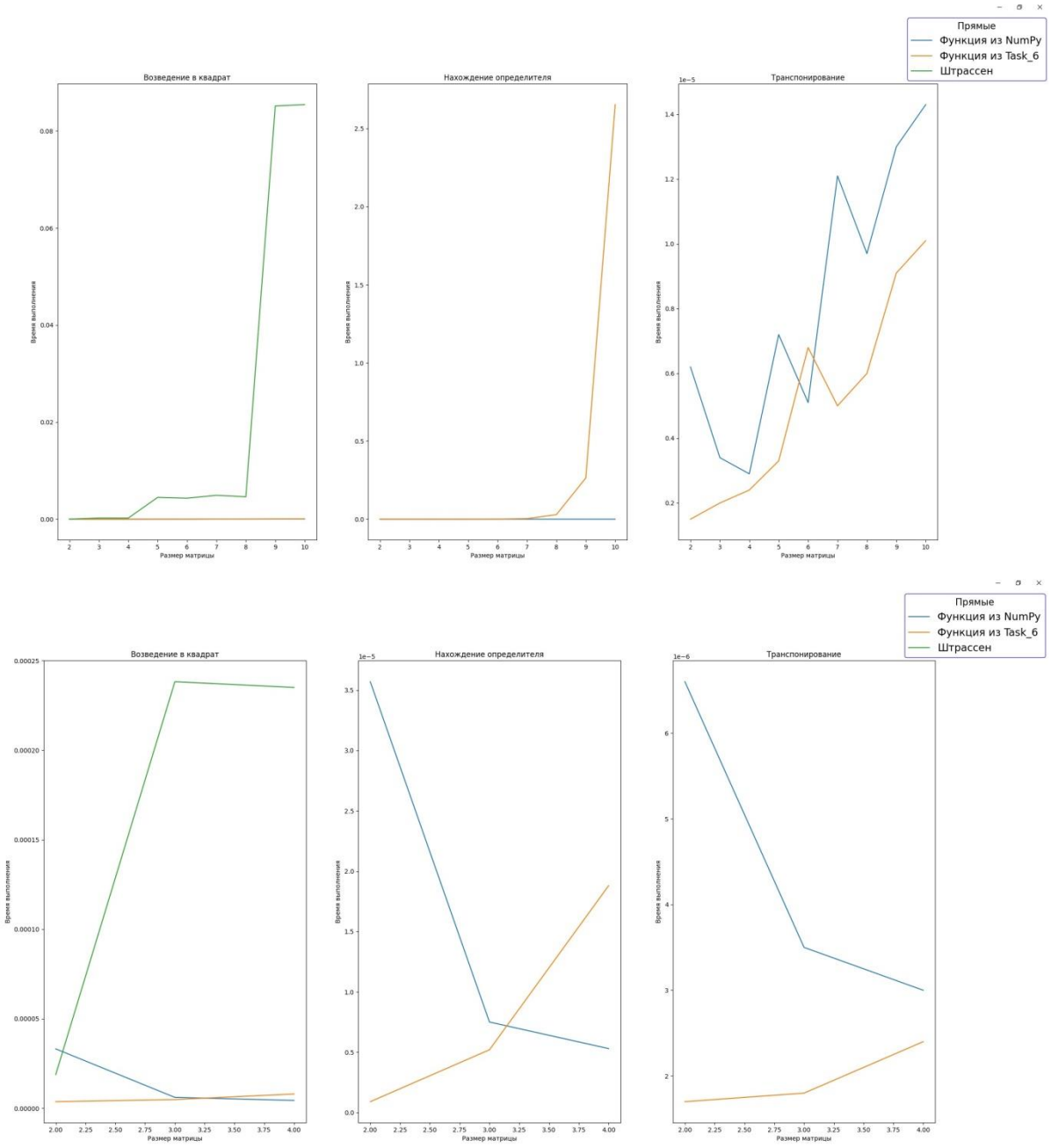
Рисунок 2. Построение графиков.

```python
def strassen(array1, array2):
    if len(array1) == 1:
        return [[array1[0][0] * array2[0][0]]]
    dar1 = divide(array1)
    dar2 = divide(array2)

    p1 = concot([strassen(dar1[0], dar2[1]), strassen(dar1[0], dar2[3])], [1, -1])
    p2 = concot([strassen(dar1[0], dar2[3]), strassen(dar1[1], dar2[3])], [1, 1])
    p3 = concot([strassen(dar1[2], dar2[0]), strassen(dar1[3], dar2[0])], [1, 1])
    p4 = concot([strassen(dar1[3], dar2[2]), strassen(dar1[3], dar2[0])], [1, -1])
    p5 = concot([strassen(dar1[0], dar2[0]), strassen(dar1[0], dar2[3]), strassen(dar1[3], dar2[0]),
                 strassen(dar1[3], dar2[3])], [1, 1, 1, 1])
    p6 = concot([strassen(dar1[1], dar2[2]), strassen(dar1[1], dar2[3]), strassen(dar1[3], dar2[2]),
                 strassen(dar1[3], dar2[3])], [1, 1, -1, -1])
    p7 = concot([strassen(dar1[0], dar2[0]), strassen(dar1[0], dar2[1]), strassen(dar1[2], dar2[0]),
                 strassen(dar1[2], dar2[1])], [1, 1, -1, -1])
    c1 = concot([p5, p4, p2, p6], [1, 1, -1, 1])
    c2 = concot([p1, p2], [1, 1])
    c3 = concot([p3, p4], [1, 1])
    c4 = concot([p5, p1, p3, p7], [1, 1, -1, -1])
    return fuse(c1, c2, c3, c4)


def divide(array):
    flag = False
    if len(array) % 2 == 1:
        line = [0]
        for i in range(len(array)):
            line.append(0)
            array[i].append(0)
        array.append(line)
        flag = True
    a1 = []
    a2 = []
    a3 = []
    a4 = []
    for i in range(len(array) // 2):
        a1.append(array[i][:len(array) // 2])
        a2.append(array[i][len(array) // 2:])
        a3.append(array[i + len(array) // 2][:len(array) // 2])
        a4.append(array[i + len(array) // 2][len(array) // 2:])
    return [a1, a2, a3, a4, flag]
```

```python
def fuse(c1, c2, c3, c4):
    fusear1 = []
    fusear2 = []
    for i in range(len(c1)):
        fusear1.append(c1[i] + c2[i])
        fusear2.append(c3[i] + c4[i])
    return fusear1 + fusear2


def concot(args, kwargs):
    size = len(args[0])
    sum = []
    for i in range(size):
        line = []
        for j in range(size):
            s = 0
            for k in range(len(args)):
                s += args[k][i][j] * kwargs[k]
            line.append(s)
        sum.append(line)
    return sum
```

Рисунки 3,4. Реализация алгоритма Штрассена.

Результат выполнения программы:



Рисунки 5,6. Результат выполнения программы, график зависимости времени работы функций в зависимости от размеров квадратичных матриц.

Вывод:

Как видно из графиков, наша реализация алгоритма Штрассена работает медленнее даже чем произведение матриц сложностью O(n^3). Нам кажется, что это связано с константой, которая проявляется из-за различных сложных операций по типу if-ов. Однако на матрицах с очень большими размерами (больше 100x100) константа станет пренебрежимо мала, и алгоритм Штрассена будет работать быстрее по сравнению с произведением матриц сложностью O(n^3).

Выполняя данное задание мы освоили алгоритм Штрассена и улучшили свои навыки использования библиотеки numpy.

Задание 2

Описание задания:

Создать таблицу nxn, заполненную случайными величинами в диапазоне от 1 до 30. Удалить значения из 10 случайных ячеек. Реализовать алгоритм, восстанавливающий данные путем линейной аппроксимации. Реализовать алгоритм, восстанавливающий значения путем корреляционного восстановления. Коррелируемые между собой ряды измерений выбирает пользователь при запуске программы. Проанализировать достоинства и недостатки методов восстановления данных.

Листинг (Решение):

Программа создаёт таблицу nxn в google sheets (подробное описание см. в задании номер 3) (рисунок 7)

```
import random,gspread,time,numpy as np
def returner(x1,x2,k,b):
    for x in range(x2,x1,-1):
        cell[x] = k*x + b


sa = gspread.service_account(filename="Token.json")
sheet = sa.open("random_values")
worksheet = sheet.worksheet("1")

n = int(input("Введите n: "))
worksheet.clear()
if worksheet.col_count < n:
    worksheet.add_cols(n-worksheet.col_count)
if worksheet.row_count < n:
    worksheet.add_rows(n-worksheet.col_count)
minimum = 1
maximum = 30
```

Рисунок 7. Создание таблицы и функция returner.

После этого из таблицы удаляется 10 значений (они становятся пустыми) (рисунок 8).

```
while count != 10:
    a = random.randint(1,n)
    b = random.randint(1,n)
    if cells[a-1][b-1] != "":
        worksheet.update_cell(a,b,"")
        cells[a-1][b-1] = ""
        count += 1
        time.sleep(1.1)

for i in range(len(cells)):
    for j in cells[i]:
        cell.append(j)
```

Рисунок 8. Удаление элементов.

Далее таблица приводится к виду одномерного массива. Линейная аппроксимация берёт за значения по x – индексы ячеек, а значения в ячейках – за y. Между двумя соседними точками с существующими значениями “строится” прямая (вычисляются коэффициенты для её уравнения.) Таким образом пустые точки, лежащие между 2 данными заполняются(вычисляются их координаты по y с помощью значения их индекса) (рисунки 9, 10).

```
def aproximation():
    flag = False
    count = 0
    for x in range(len(cell)):
        if cell[x] != "" and not flag:
            y1,x1 = int(cell[x]),x
            flag = True
            count += 1
        elif cell[x] != "" and flag:
            y2,x2 = int(cell[x]),x
            k = (y2-y1)/(x2-x1)
            b = y1 - k*x1
            returner(x1,x2,k,b)
            y1,x1 = y2,x2
            count+=1
    if count <= 1:
        print("Восстановить невозможно")
else:
    if cell[0] == "":
        for i in range(len(cell)):
            if cell[i] != "":
                cell[i] = float(cell[i])
                y1, x1 = cell[i], i
                y2,x2 = cell[x1+1], x1 + 1
                break
        k = (y2 - y1) / (x2 - x1)
        b = y1 - k * x1
        returner(-1,x1-1,k,b)
    if cell[-1] == "":
        for i in range(len(cell)-1,-1,-1):
            if cell[i] != "":
                cell[i] = float(cell[i])
                y1,x1 = cell[i],i
                y2,x2 = cell[x1-1], x1 - 1
                break
        k = (y2 - y1) / (x2 - x1)
        b = y1 - k * x1
        returner(x1,len(cell)-1,k,b)
```

Рисунки 9,10. Нахождение коэффициентов.

Корреляционное восстановление: Пользователь вводит номер строки, которую хочет восстановить. Далее находится строчка из таблицы, у которой наибольшей коэффициент корреляции с восстанавливаемой строчкой. С помощью данных в этой строчке по методу наименьших квадратов и восстанавливаются значения (Рисунки 11, 12, 13, 14, 15).

```
def correlation():
    raw = int(input("Введите коррелируемую строку: ")) - 1
    corr_line = []
    coeffs = {}
    lines ={}
    for i in cells[raw]:
        corr_line.append(i)
    for i in range(len(cells)):
        if i != raw:
            corr_line_up = []
            line = []
            flag = True
            for j in range(len(cells[i])):
                if corr_line[j] == "" and cells[i][j] == "":
                    flag = False
                    break
                break
            if corr_line[j] != "" and cells[i][j] != "":
                corr_line_up.append(int(corr_line[j]))
                line.append(int(cells[i][j]))
            if flag:
                if len(corr_line_up) > 2:
                    coeffs[i] = regcoef(corr_line_up,line)
                    lines[i] = [line,corr_line_up]
    max = 0
    maxindex = 0
    for key in coeffs:
        if abs(coeffs[key]) > max:
            max = coeffs[key]
            maxindex = key
```

```
if len(lines) > 0:
    a , b = little_squares(lines[maxindex][0],lines[maxindex][1])
    for i in range(len(corr_line)):
        if corr_line[i] == "":
            x = int(cells[maxindex][i])
            corr_line[i] = a * x + b
else:
    print("Невозможно посчитать")
return corr_line,raw
```

```
def little_squares(array1,array2):
    sumxy = 0
    sumx = 0
    sumy = 0
    sumx2 = 0
    for i in range(len(array1)):
        sumxy += array1[i]*array2[i]
        sumx += array1[i]
        sumy += array2[i]
        sumx2 += array1[i]**2
    a = (len(array1) * sumxy - sumx*sumy)/(len(array1)*sumx2-(sumx)**2)
    b = (sumy - a*sumx)/len(array1)
    return a,b
```

```
def regcoef(array1,array2):
    middlex = 0
    middley = 0
    fluctsumxy = 0
    fluctsumx = 0
    fluctsumy = 0
    for i in array1:
        middlex += i
    for i in array2:
        middley += i
    middlex = middlex / len(array1)
    middley = middley / len(array2)
    for i in range(len(array1)):
        fluctsumxy += (array1[i] - middlex)*(array2[i] - middley)
        fluctsumx += (array1[i] - middlex)**2
        fluctsumy += (array2[i] - middley) ** 2
    rxy = fluctsumxy/((fluctsumx*fluctsumy)**(1/2))
    return rxy
```

Рисунки 11, 12, 13, 14, 15 – функции, нужные для корреляционного восстановления.

Результат выполнения программы:

A	B	C	D
11	11	6	12
4	19	14	2
23	11	19	15
7	2	3	28

4	19		
23		19	15
		3	

A	B	C	D
-56	-41	-26	-11
4	19	20,33333333	21,66666667
23	21	19	15
11	7	3	-1

Рисунки 16,17,18. Изменение таблицы в процессе выполнения программы и конечный результат (аппроксимация).

A	B	C	D	E	F	G	H
14	27	24	23	10	1	11	15
6	13	19	10	15		8	24
	15	10	28	20	20		13
21	13	27		8	18	28	27
9	13	13	22	14	19	25	25
	13		15	9		22	25
18	22	26	27	20	26	4	27
9	28	25	28,78873239	16,56338028	26	-11,38028169	28

Рисунок 19. Результат корреляционного восстановления.

Вывод:
В результате выполнения задания познакомились с корреляционным восстановлением и линейной аппроксимацией. Получили новые знания о восстановлении данных.

Задание 3

Описание задания:

Создать таблицу nxn, заполненную случайными величинами в диапазоне от 1 до 30. Реализовать алгоритм поиска математического ожидания и дисперсии для каждого из рядов созданной таблицы.

Листинг (Решение):

Для решения поставленной задачи (Рисунок 20) был задействован google sheets api, с помощью модуля gspread. Сама программа заполняла случайными числами таблицу в google sheets, а затем выполняла алгоритм для вычисления математического ожидания и дисперсии для каждого ряда таблицы.

```
import random, gspread, time

sa = gspread.service_account(filename="Token.json")
sheet = sa.open("random_values")
worksheet = sheet.worksheet("1")

n = int(input("Введите n: "))
worksheet.clear()
if worksheet.col_count < n:
    worksheet.add_cols(n-worksheet.col_count)
if worksheet.row_count < n:
    worksheet.add_rows(n-worksheet.col_count)
minimum = 1
maximum = 30
for i in range(n):
    for j in range(n):
        worksheet.update_cell(i+1, j+1, random.randint(minimum, maximum))
        time.sleep(1.1)


normal_math_waiting = ((minimum + maximum) * (maximum / 2)) / maximum
normal_dispers = 0
cells = worksheet.get_all_values()
for i in range(minimum, maximum+1):
    normal_dispers += normal_dispers + (normal_math_waiting - i)**2
normal_dispers = normal_dispers / maximum
for i in range(n):
    dispers = 0
    summa = sum(map(int, cells[i])) / n
    for j in range(n):
        dispers = dispers + (int(cells[i][j]) - summa)**2
    dispers = dispers / n
    print(f"Математическое ожидание {i+1} строки: {summa}")
    print(f"Дисперсия для {i+1} строки: {dispers}")

print(f"Ожидаемое математическое ожидание: {normal_math_waiting}")
print(f"Ожидаемая дисперсия: {normal_dispers}")
```

Рисунок 20. Основной код программы.

Результат выполнения программы:

```
Введите n: 5
Математическое ожидание 1 строки: 12.0
Дисперсия для 1 строки: 88.4
Математическое ожидание 2 строки: 19.6
Дисперсия для 2 строки: 52.239999999999995
Математическое ожидание 3 строки: 17.6
Дисперсия для 3 строки: 42.24000000000001
Математическое ожидание 4 строки: 17.4
Дисперсия для 4 строки: 68.24000000000001
Математическое ожидание 5 строки: 11.0
Дисперсия для 5 строки: 17.2
Ожидаемое математическое ожидание: 15.5
Ожидаемая дисперсия: 6594564359.925

Process finished with exit code 0
```

A	B	C	D	E
23	12	1	2	22
20	17	23	30	8
29	15	17	18	9
23	1	21	20	22
8	19	11	9	8

Рисунки 21,22. Результаты вычислений и заполненная таблица.

Вывод:
Работая над выполнением данного задания мы узнали, что такое математическое ожидание, дисперсия и как их вычислять.