Лабораторная работа No1

«Подготовка вспомогательных средств разработки»

Цель работы

Подготовить библиотеки для выполнения операций с векторами, матрицами, обработкой, хранением и представлением данных.

Задание

1. Реализовать вспомогательную библиотеку для вычисления скалярного произведения, умножение матрицы на вектор, матрицы на матрицу.
2. Проверить корректность работы с помощью сторонних библиотек.
3. Реализовать оценку погрешности вычисления шара при различных значениях корней из 2.
4. Реализовать вывод графиков произвольных двумерных функций.

Реализация

1. Были составлены 2 программы, реализующие матрицы и скалярное произведение
2. была произведена оценка работы путем сравнения с numpy
3. Была посчитана относительная погрешность вычисления шара при различных значениях корней из 2, а так же с помошью matplotlib нарисованы 2 графика при разных значениях корня
4. Была реализована функция с помощью matplotlib для вывода графиков произвольных двумерных функций

my_matrix.py

```
class MyMatrix(object):

    def __init__(self, matrix):
        self.matrix = matrix

    def mult_on_vector(self, vector):

        multed_matrix = []

        for i in range(len(self.matrix)):
```

```
            vector_elem = 0

            for j in range(len(vector)):
                # print("a[{},{}]*b[{}]: {}*{} +  ".format(j+1,i+1,j+1,self.matrix[i][j
                vector_elem += self.matrix[i][j] * vector[j]

            multed_matrix.append(vector_elem)

        return multed_matrix

    def mult_on_matrix(self, m):

        multed_matrix = []

        if len(m.matrix) != len(self.matrix[0]):

            print("Ошибка в перемножении матриц")

        else:

            n_row_matrix1 = len(self.matrix)
            n_column_matrix1 = len(self.matrix[0])
            n_row_matrix2 = n_column_matrix1
            n_column_matrix2 = len(m.matrix[0])

            for i in range(0, n_row_matrix1):

                temp_matrix = []

                for j in range(0, n_column_matrix2):

                    sum = 0

                    for k in range(0, n_column_matrix1):
                        sum += self.matrix[i][k] * m.matrix[k][j]

                    temp_matrix.append(sum)

                multed_matrix.append(temp_matrix)

        return multed_matrix
```

my_scalar

```
def scalar_multiply(vector1, vector2):

    if len(vector1) > len(vector2):
        length = len(vector1)
    else:
        length = len(vector2)

    scalar_digit = 0
```

```
    for i in range(0, length):
        scalar_digit += vector1[i]*vector2[i]

    return scalar_digit
```

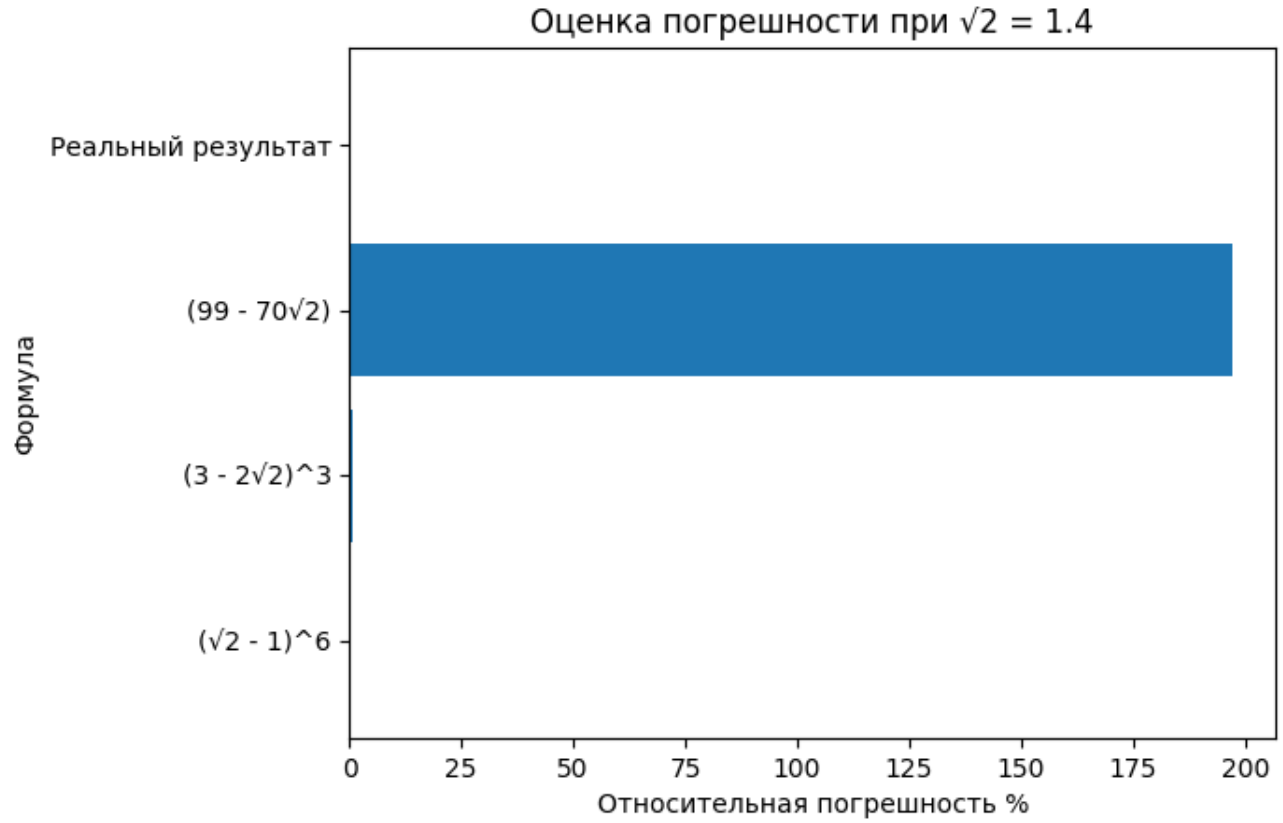
main.py

```
import numpy as np
import matplotlib.pyplot as plt

from my_python_math.my_matrix import MyMatrix
from my_python_math.my_scalar import scalar_multiply


def test(sc_vector1, sc_vector2, my_matrix, vector_for_matrix, matrix_for_my_matrix):

    nump_sc_vector1 = np.array(sc_vector1)
    nump_sc_vector2 = np.array(sc_vector2)

    print("numpy version: {}".format(np.dot(nump_sc_vector1, nump_sc_vector2)))
    print("my version:    {}".format(scalar_multiply(sc_vector1, sc_vector2)))

    nump_my_matrix = np.array(my_matrix)
    nump_vector_for_matrix = np.array(vector_for_matrix)
    my_matrix_class = MyMatrix(my_matrix)

    print("numpy version multiply on vector: {}".format(np.matmul(nump_my_matrix, nump_
    print("my version multiply on vector:    {}".format(my_matrix_class.mult_on_vector(

    matrix_for_my_matrix_class = MyMatrix(matrix_for_my_matrix)
    nump_matrix_for_my_matrix = np.array(matrix_for_my_matrix)

    print("numpy version multiply on matrix: {}".format(np.dot(nump_my_matrix, nump_mat
    print("my version multiply on matrix:    {}".format(my_matrix_class.mult_on_matrix(


def error_calculation(x):

    real_result = 0.005051

    formula_power_of_six = (x - 1)**6
    formula_power_of_three = (3 - 2*x)**3
    formula_power_of_one = 99 - 70*x

    print("Погрешность первой формулы при √2 = {}:  {}".format(x, abs(formula_power_of_
    print("Погрешность второй формулы при √2 = {}:  {}".format(x, abs(formula_power_of_
    print("Погрешность третьей формулы при √2 = {}: {}".format(x, abs(formula_power_of_
    print()

    y_labels = ["(√2 - 1)^6", "(3 - 2√2)^3", "(99 - 70√2)", "Реальный результат"]

    x_labels = [abs(formula_power_of_six - real_result)/real_result,
```

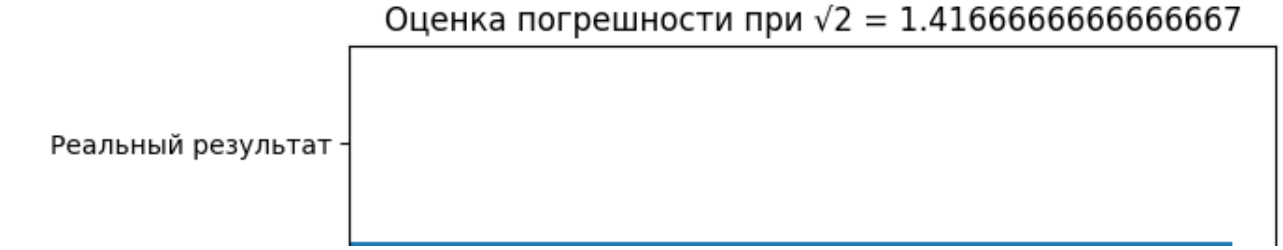
```
            abs(formula_power_of_three - real_result)/real_result,
            abs(formula_power_of_one - real_result)/real_result,
            real_result]

        plt.rcdefaults()

        plt.barh(y_labels, x_labels)
        plt.ylabel("Формула")
        plt.xlabel('Относительная погрешность %')
        plt.title('Оценка погрешности при √2 = {}'.format(x))

        plt.show()
        print()


    def draw_chart(x, y):
        x = np.array(x)
        y = np.array(y)
        plt.plot(x, y)
        plt.title("График двумерной функции")
        plt.show()


    if __name__ == '__main__':

        # Реализовать вспомогательную библиотеку для вычисления скалярного произведения,
        # умножение матрицы на вектор, матрицы наматрицу.
        # Проверить корректность работы с помощью сторонних библиотек.
        vector11 = [1, 2, 3, 4, 5]
        vector12 = [5, 4, 3, 2, 1]
        m11 = [[1, 2, 3],[3, 2, 1],[4, 5, 6]]
        vector13 = [7, 8, 9]
        m12 = [[1, 2, 3],[3, 2, 1],[4, 5, 6]]
        test(vector11, vector12, m11, vector13, m12)

        print()

        vector21 = [1, 2, 3, 4, 5, 9, 1, 142]
        vector22 = [5, 4, 3, 2, 1, 7, 78, 35]
        m21 = [[1, 2, 3, 4], [3, 2, 1, 1], [4, 5, 6, 7], [7, 66, 4, 6]]
        vector23 = [7, 8, 9, 123]
        m22 = [[1, 2, 3, 1], [3, 2, 1, 1], [4, 5, 6, 1], [505, 1, 6, 1]]
        test(vector21, vector22, m21, vector23, m22)

        print()

        # Реализовать оценку погрешности вычисления шара при различных значениях корней из
        error_calculation(7/5)
        error_calculation(17/12)

        print()

        # Реализовать вывод графиков произвольных двумерных функций.
        x = [4, 5, 6, 7, 8]
        y = [1, 2, -6, 0, 4]
```

```
    draw_chart(x, y)
```

```
numpy version: 35
my version:    35
numpy version multiply on vector: [ 50  46 122]
my version multiply on vector:    [50, 46, 122]
numpy version multiply on matrix: [[19 21 23]
 [13 15 17]
 [43 48 53]]
my version multiply on matrix:    [[19, 21, 23], [13, 15, 17], [43, 48, 53]]

numpy version: 5146
my version:    5146
numpy version multiply on vector: [ 542  169  983 1351]
my version multiply on vector:    [542, 169, 983, 1351]
numpy version multiply on matrix: [[2039   25   47   10]
 [ 518   16   23    7]
 [3578   55   95   22]
 [3251  172  147   83]]
my version multiply on matrix:    [[2039, 25, 47, 10], [518, 16, 23, 7], [3578, 5

Погрешность первой формулы при √2 = 1.4:  0.18907147099584345
Погрешность второй формулы при √2 = 1.4:  0.5838447832112494
Погрешность третьей формулы при √2 = 1.4: 196.98059790140567
```




```
Погрешность первой формулы при √2 = 1.4166666666666667:  0.03598908842427385
Погрешность второй формулы при √2 = 1.4166666666666667:  0.08342315786386512
Погрешность третьей формулы при √2 = 1.4166666666666667: 33.99676631690188
```



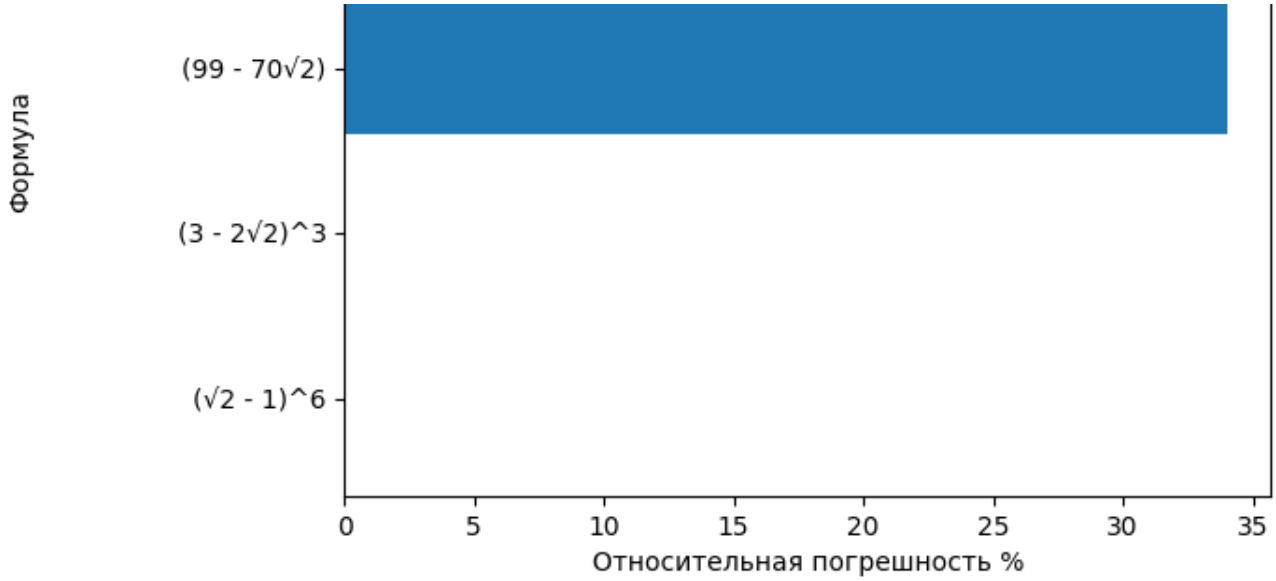


График двумерной функции