Импорт библиотек

import numpy as np

import matplotlib.pyplot as plt

Задача 1. Дан набор из p матриц размерностью (n,n) и p векторов размерностью (n,1), найти сумму произведений матриц на векторы. Написать тесты для кода

def sum\_prod(X, V):

'''

X - матрицы (n, n)

V - векторы (n, 1)

Гарантируется, что len(X) == len(V)

'''

def sum\_matrix\_vector\_products(matrices, vectors):

result = 0

for matrix, vector in zip(matrices, vectors):

result += np.dot(matrix, vector).sum()

return result

# Тесты

def test\_sum\_matrix\_vector\_products():

matrix1 = np.array([[1, 2], [3, 4]])

matrix2 = np.array([[5, 6], [7, 8]])

vector1 = np.array([[1], [2]])

vector2 = np.array([[3], [4]])

assert sum\_matrix\_vector\_products([matrix1, matrix2], [vector1, vector2]) == 70

matrix3 = np.array([[2, 3], [4, 5]])

vector3 = np.array([[2], [3]])

assert sum\_matrix\_vector\_products([matrix1, matrix2, matrix3], [vector1, vector2, vector3]) == 112

print("Все тесты пройдены успешно!")

test\_sum\_matrix\_vector\_products()

Задача 2. Дана матрица M, напишите функцию, которая бинаризует матрицу по некоторому threshold (то есть, все значения большие threshold становятся равными 1, иначе 0). Напишите тесты для кода

def binarize(M, threshold=0.5):

def binarize\_matrix(matrix, threshold):

binary\_matrix = np.where(matrix > threshold, 1, 0)

return binary\_matrix

# Тесты

def test\_binarize\_matrix():

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

threshold = 4

expected\_result = np.array([[0, 0, 0], [1, 1, 1], [1, 1, 1]])

assert np.array\_equal(binarize\_matrix(matrix, threshold), expected\_result)

matrix2 = np.array([[0.5, 1.5], [2.5, 3.5]])

threshold2 = 2

expected\_result2 = np.array([[0, 0], [1, 1]])

assert np.array\_equal(binarize\_matrix(matrix2, threshold2), expected\_result2)

print("Все тесты пройдены успешно!")

test\_binarize\_matrix()

pass

Задача 3. Напишите функцию, которая возвращает уникальные элементы из каждой строки матрицы. Напишите такую же функцию, но для столбцов. Напишите тесты для кода

def unique\_rows(mat):

# Your code goes here

pass

def unique\_columns(mat):

def unique\_elements\_in\_rows(matrix):

unique\_elements = [np.unique(row) for row in matrix]

return unique\_elements

def unique\_elements\_in\_columns(matrix):

unique\_elements = [np.unique(column) for column in matrix.T]

return unique\_elements

# Тесты

def test\_unique\_elements\_in\_rows():

matrix = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])

expected\_result = [np.array([1, 2, 3]), np.array([2, 3, 4]), np.array([3, 4, 5])]

assert all(np.array\_equal(row, expected\_row) for row, expected\_row in zip(unique\_elements\_in\_rows(matrix), expected\_result))

def test\_unique\_elements\_in\_columns():

matrix = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])

expected\_result = [np.array([1, 2, 3]), np.array([2, 3, 4]), np.array([3, 4, 5])]

assert all(np.array\_equal(column, expected\_column) for column, expected\_column in zip(unique\_elements\_in\_columns(matrix), expected\_result))

print("Все тесты пройдены успешно!")

test\_unique\_elements\_in\_rows()

test\_unique\_elements\_in\_columns()

Задача 4. Напишите функцию, которая заполняет матрицу с размерами (m,n)

случайными числами, распределенными по нормальному закону. Затем считает мат. ожидание и дисперсию для каждого из столбцов и строк, а также строит для каждой строки и столбца гистограмму значений (использовать функцию hist из модуля matplotlib.plot)

def generate\_normal\_matrix\_and\_plot\_statistics(m, n):

# Генерация матрицы с случайными числами

matrix = np.random.randn(m, n)

# Вычисление мат. ожидания и дисперсии для строк и столбцов

row\_means = np.mean(matrix, axis=1)

row\_variances = np.var(matrix, axis=1)

column\_means = np.mean(matrix, axis=0)

column\_variances = np.var(matrix, axis=0)

# Построение гистограмм значений для каждой строки

for i in range(m):

plt.hist(matrix[i], bins=10)

plt.title(f"Histogram of values for row {i}")

plt.show()

# Построение гистограмм значений для каждого столбца

for j in range(n):

plt.hist(matrix[:,j], bins=10)

plt.title(f"Histogram of values for column {j}")

plt.show()

return row\_means, row\_variances, column\_means, column\_variances

# Пример использования функции

m = 5 # количество строк

n = 3 # количество столбцов

row\_means, row\_variances, column\_means, column\_variances = generate\_normal\_matrix\_and\_plot\_statistics(m, n)

print("Мат. ожидание для каждой строки:")

print(row\_means)

print("Дисперсия для каждой строки:")

print(row\_variances)

print("Мат. ожидание для каждого столбца:")

print(column\_means)

print("Дисперсия для каждого столбца:")

print(column\_variances)

Задача 5. Напишите функцию, которая заполняет матрицу (m,n) в шахматном порядке заданными числами a и b. Напишите тесты для кода

def chess(m, n, a, b):

matrix = [[0 for \_ in range(n)] for \_ in range(m)]

current\_num = a

for i in range(m):

for j in range(n):

matrix[i][j] = current\_num

current\_num = b if current\_num == a else a

return matrix

# Тестирование функции

def chess ():

m = 4

n = 5

a = 1

b = 2

expected\_result = [[1, 2, 1, 2, 1],

[2, 1, 2, 1, 2],

[1, 2, 1, 2, 1],

[2, 1, 2, 1, 2]]

result = chess (m, n, a, b)

assert result == expected\_result, "Тест не пройден!"

print("Тест пройден успешно.")

chess()

Задача 6. Напишите функцию, которая отрисовывает прямоугольник с заданными размерами (a, b) на изображении размера (m, n), цвет фона задайте в схеме RGB, как и цвет прямоугольника. Цвета также должны быть параметрами функции. Напишите аналогичную функцию но для овала с полуосями a и b. Напишите тесты для кода. Примечание: уравнение эллипса (границы овала) можно записать как:

def draw\_rectangle(a, b, m, n, rectangle\_color, background\_color):

def draw\_rectangle(image, m, n, a, b, bg\_color=(255, 255, 255), rect\_color=(0, 0, 0)):

image[:, :] = bg\_color

start\_x = (m - a) // 2

start\_y = (n - b) // 2

end\_x = start\_x + a

end\_y = start\_y + b

image[start\_x:end\_x, start\_y:end\_y] = rect\_color

def draw\_oval(image, m, n, a, b, bg\_color=(255, 255, 255), oval\_color=(0, 0, 0)):

image[:, :] = bg\_color

center\_x = m // 2

center\_y = n // 2

for i in range(m):

for j in range(n):

if ((i - center\_x) / a)\*\*2 + ((j - center\_y) / b)\*\*2 <= 1:

image[i, j] = oval\_color

def test\_draw\_shapes():

m = 300

n = 400

a = 100

b = 150

# Создание изображения

image = np.zeros((m, n, 3), dtype=np.uint8)

# Отрисовка прямоугольника

draw\_rectangle(image, m, n, a, b, bg\_color=(255, 255, 255), rect\_color=(0, 0, 255))

plt.imshow(image)

plt.title("Rectangle")

plt.show()

# Отрисовка овала

draw\_oval(image, m, n, a, b, bg\_color=(255, 255, 255), oval\_color=(255, 0, 0))

plt.imshow(image)

plt.title("Oval")

plt.show()

# Запуск тестов

test\_draw\_shapes()

def draw\_ellipse(a, b, m, n, ellipse\_color, background\_color):

# Your code goes here

pass

Задача 7. Дан некий временной ряд. Для данного ряда нужно найти его: математическое ожидание, дисперсию, СКО, найти все локальные максимумы и минимумы (локальный максимум - это точка, которая больше своих соседних точек, а локальный минимум - это точка, которая меньше своих соседей), а также вычислить для данного ряда другой ряд, получаемый методом скользящего среднего с размером окна p.

Примечание: метод скользящего среднего подразумевает нахождение среднего из подмножетсва ряда размером

# Ваш временной ряд данных

time\_series = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]

# Математическое ожидание

mean = np.mean(time\_series)

# Дисперсия

variance = np.var(time\_series)

# Стандартное отклонение (СКО)

std\_deviation = np.sqrt(variance)

# Локальные максимумы и минимумы

local\_max = [time\_series[i] for i in range(1, len(time\_series)-1) if time\_series[i] > time\_series[i-1] and time\_series[i] > time\_series[i+1]]

local\_min = [time\_series[i] for i in range(1, len(time\_series)-1) if time\_series[i] < time\_series[i-1] and time\_series[i] < time\_series[i+1]]

# Скользящее среднее

def sliding\_mean(data, window\_size):

return [np.mean(data[i:i+window\_size]) for i in range(len(data) - window\_size + 1]

# Размер окна для скользящего среднего

p = 3

smoothed\_series = sliding\_mean(time\_series, p)

# Вывод полученных результатов

print("Математическое ожидание:", mean)

print("Дисперсия:", variance)

print("СКО:", std\_deviation)

print("Локальные максимумы:", local\_max)

print("Локальные минимумы:", local\_min)

print("Скользящее среднее с окном размером", p, ":", smoothed\_series)

Задача 8. Дан некоторый вектор с целочисленными метками классов, напишите функцию, которая выполняет one-hot-encoding для данного вектора

One-hot-encoding - представление, в котором на месте метки некоторого класса стоит 1, в остальных позициях стоит 0. Например для вектора [0, 2, 3, 0] one-hot-encoding выглядит как: [[1, 0, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1], [1, 0, 0, 0]]

def one\_hot\_encode(vector, num\_classes):

encoded = np.zeros((len(vector), num\_classes)) # Создаем нулевую матрицу нужного размера

for i in range(len(vector)):

encoded[i][vector[i]] = 1 # Устанавливаем 1 в позицию метки класса

return encoded

# Пример использования

vector = [0, 2, 3, 0]

num\_classes = 4

one\_hot\_encoded = one\_hot\_encode(vector, num\_classes)

print(one\_hot\_encoded)