Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы численного анализа

**ОТЧЁТ**

к лабораторной работе

на тему

Нахождение собственных значений и собственных векторов

Выполнил: студент группы 253504

Фроленко Кирилл Юрьевич

Проверил: Анисимов Владимир Яковлевич

Минск 2023

**Содержание**

1. Цель работы
2. Теоретические сведения
3. Программная реализация
4. Решение задания
5. Тестовые примеры
6. Выводы

Вариант 28

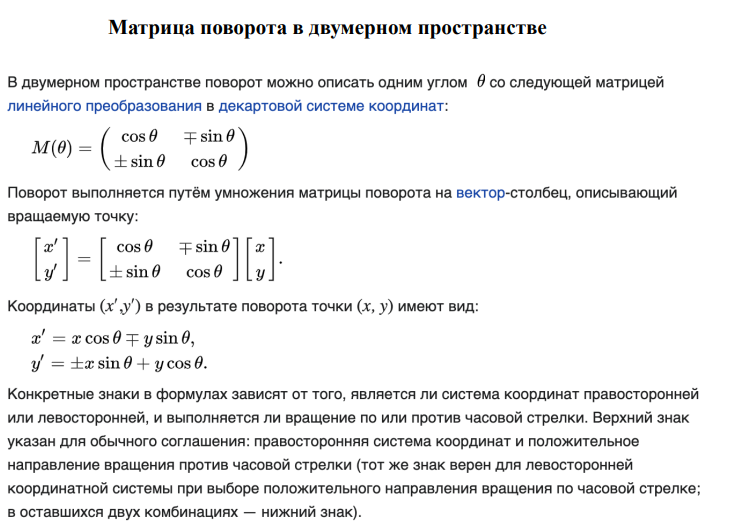
# **Цели выполнения задания**

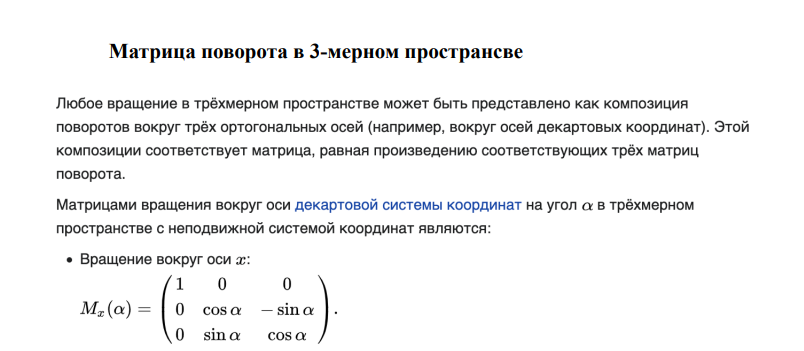
1) Рассмотреть методы вычисления собственных значений и векторов

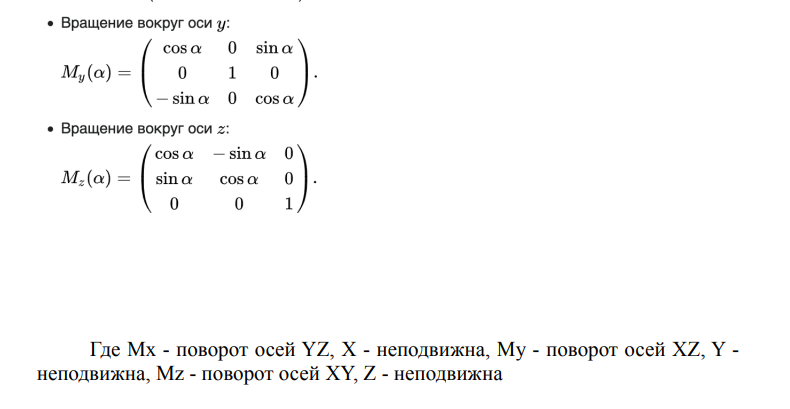
2) Реализовать методы вращений Якоби и метод Данилевского

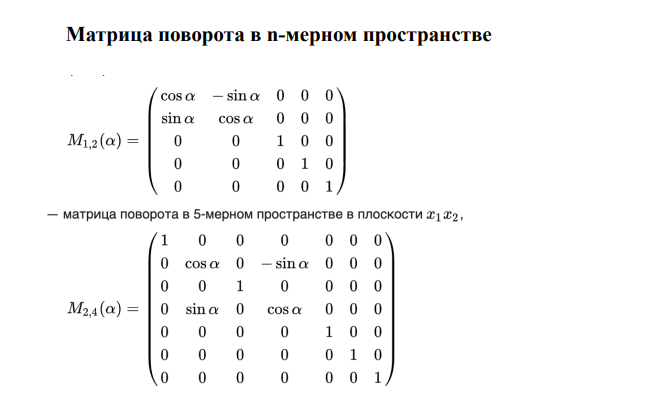
# **Краткие теоретические сведения**

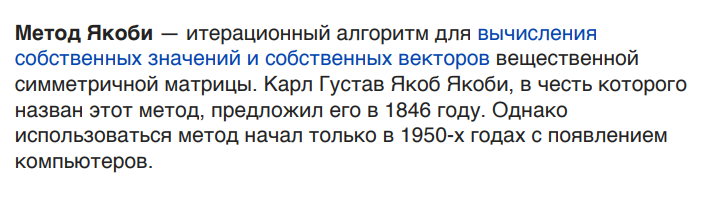
Итеративные алгоритмы решают задачу вычисления собственных значений путём построения последовательностей, сходящихся к собственным значениям. Некоторые алгоритмы дают также последовательности векторов, сходящихся к собственным векторам. Чаще всего последовательности собственных значений выражаются через последовательности подобных матриц, которые сходятся к треугольной или диагональной форме, что позволяет затем просто получить собственные значения. Последовательности собственных векторов выражаются через соответствующие матрицы подобия.

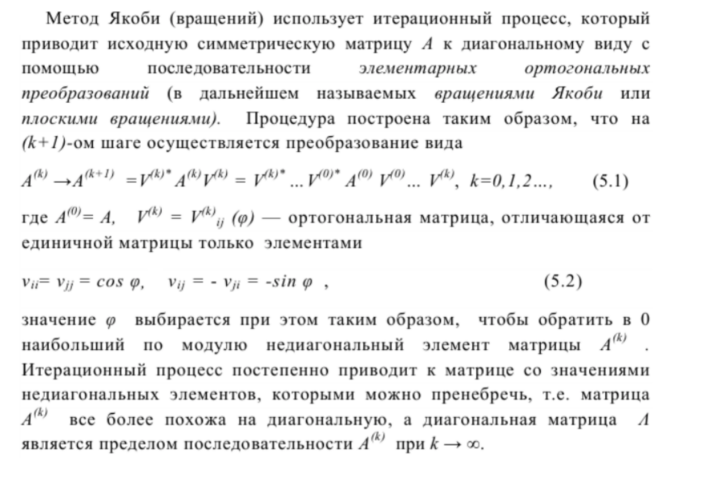


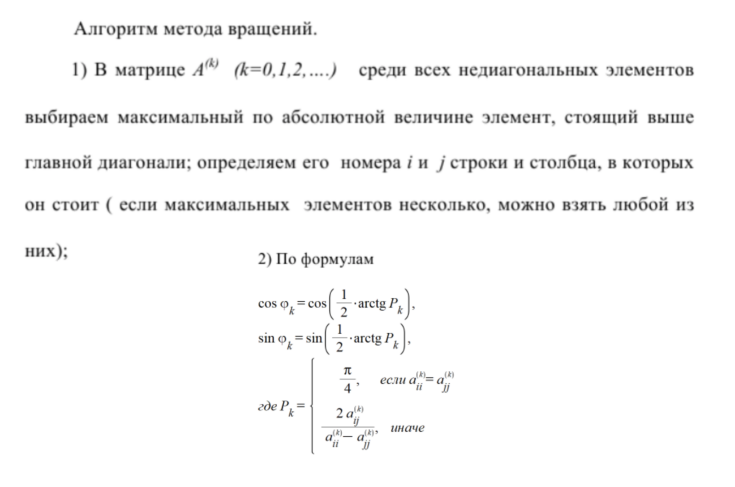


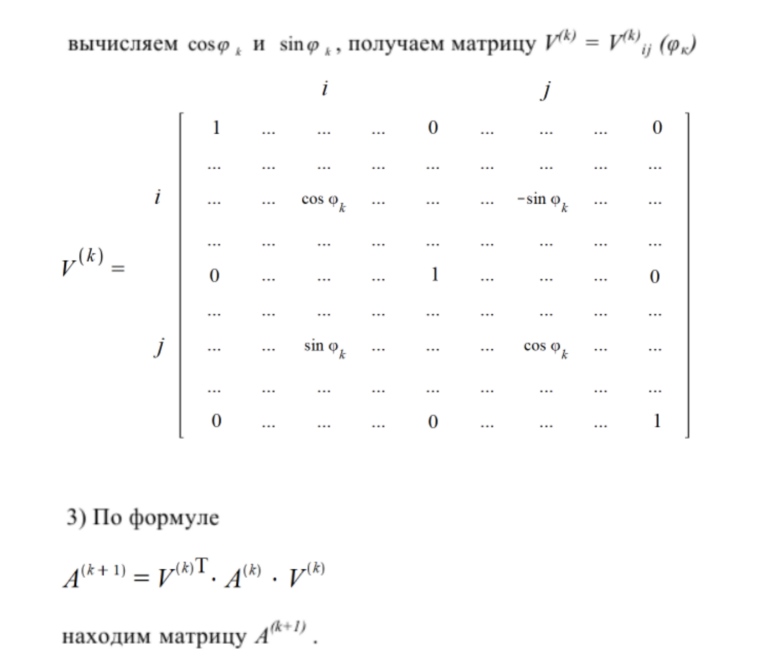


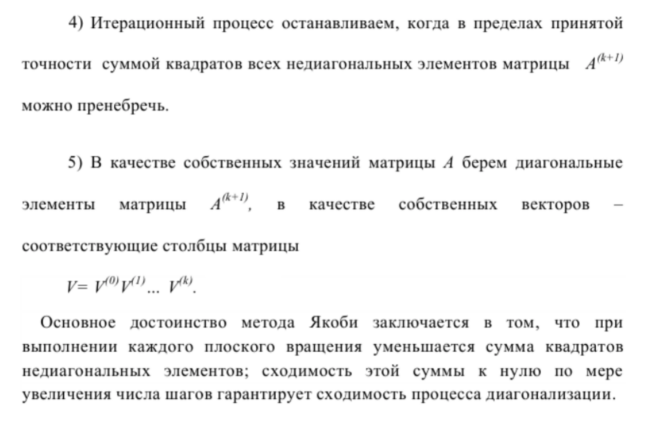


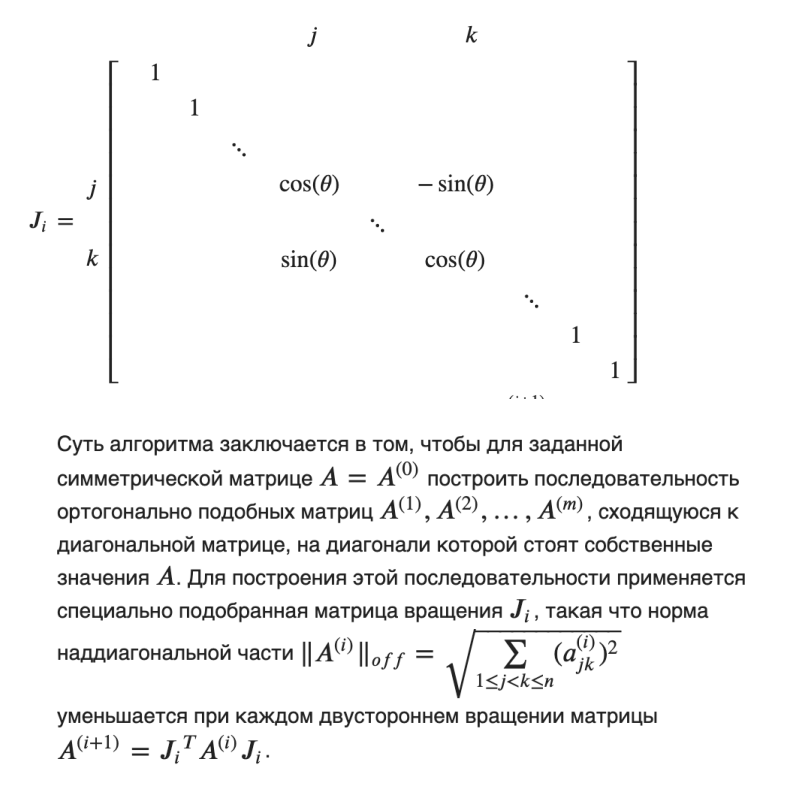
****

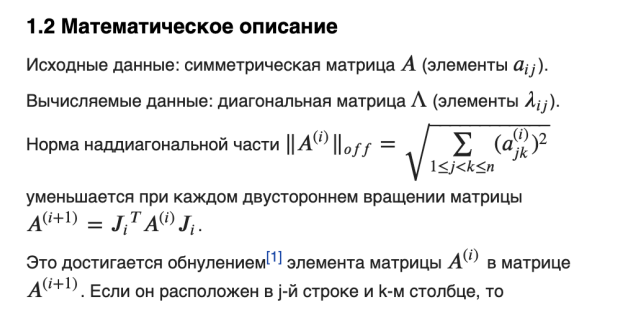
****

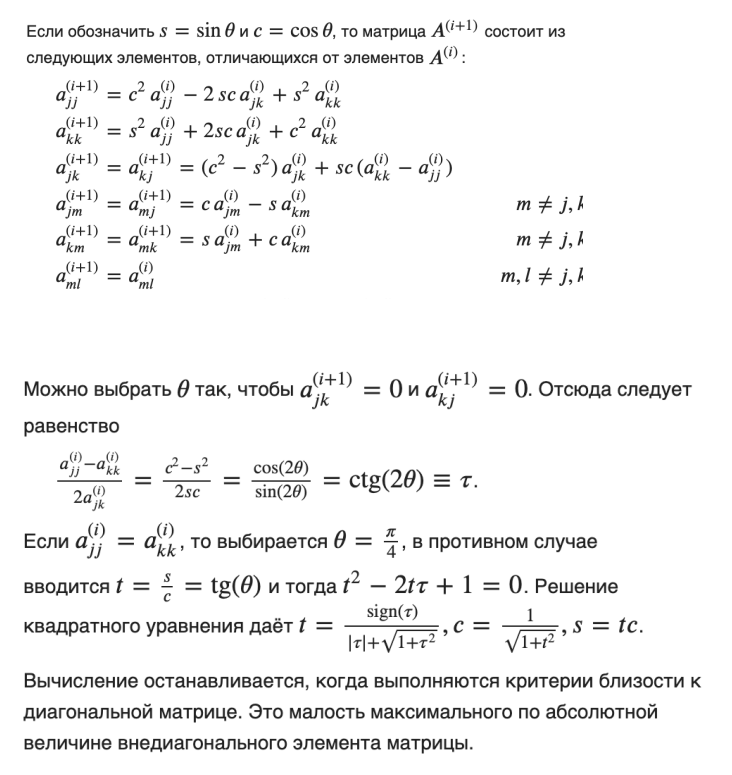
****

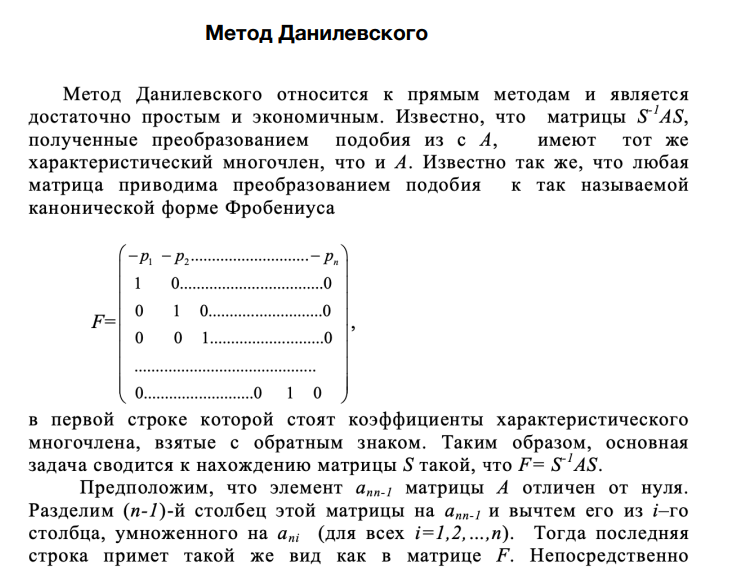
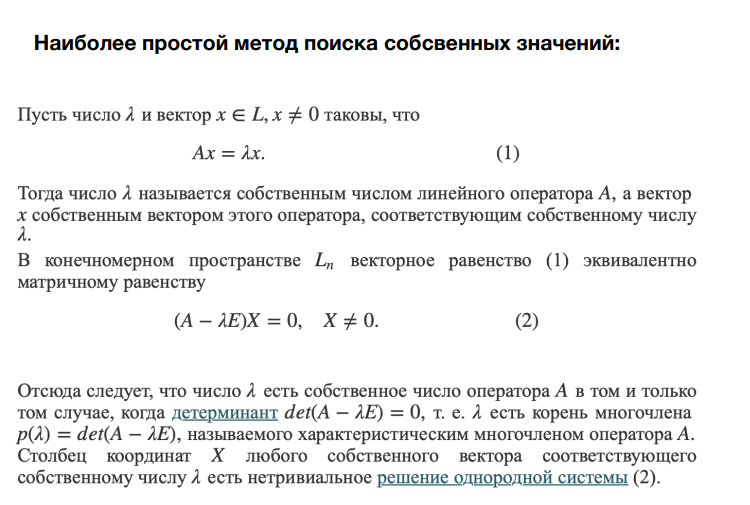
****

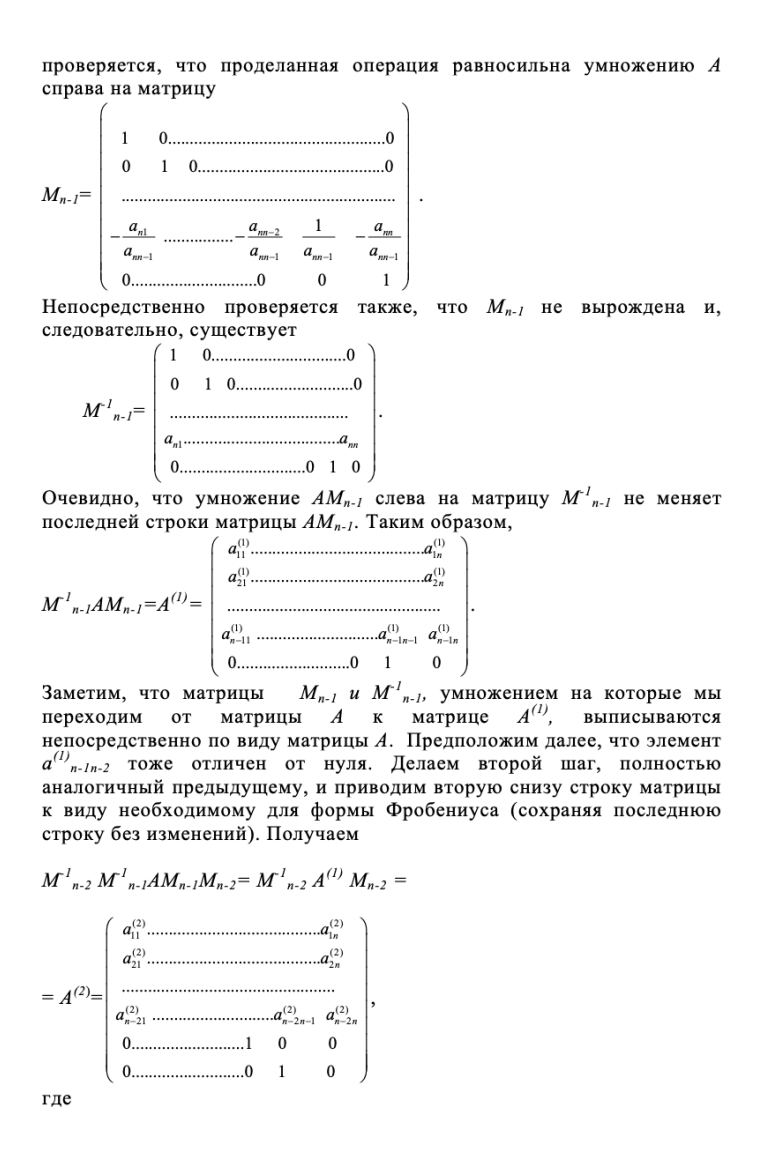
****

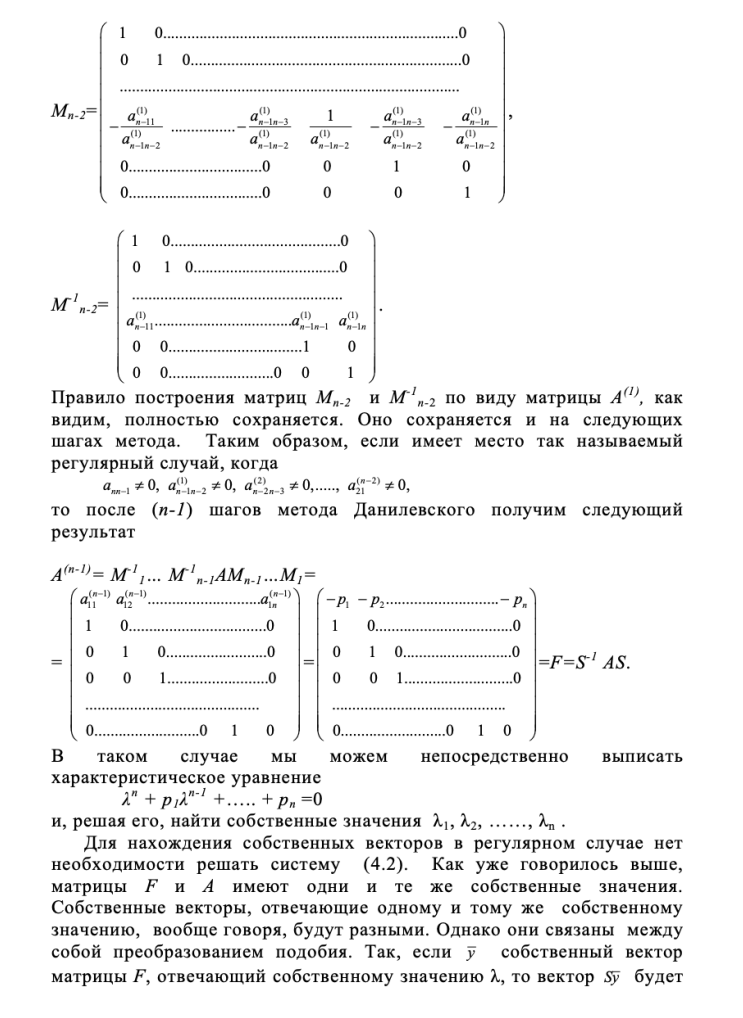
****

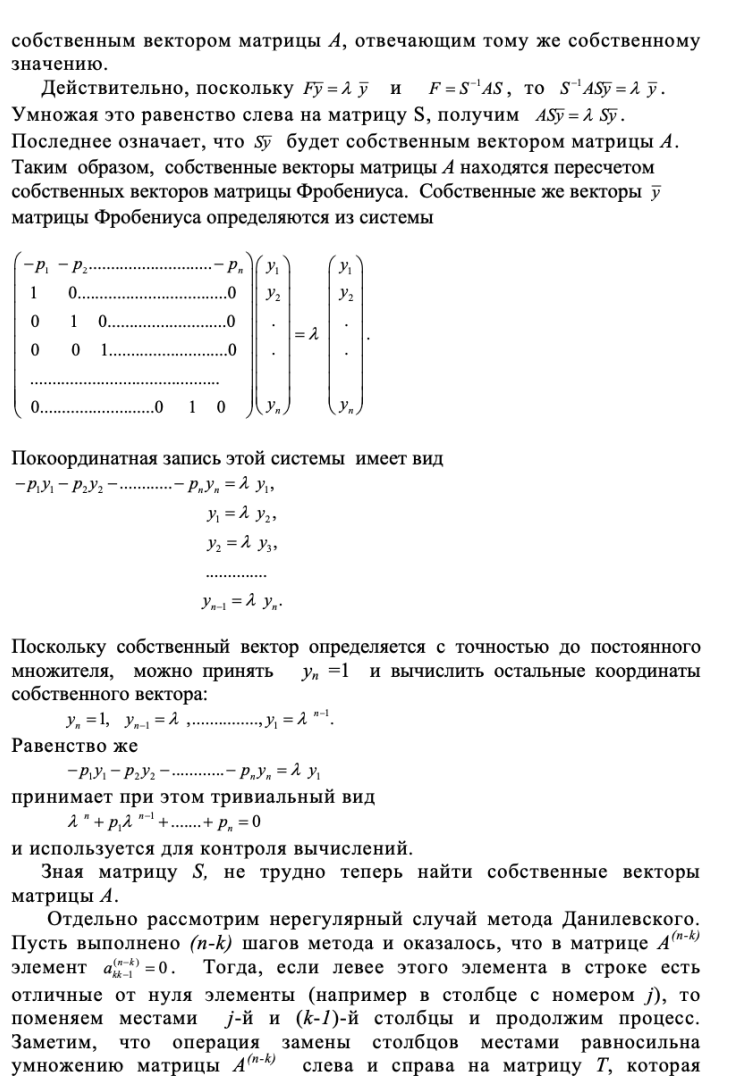
****

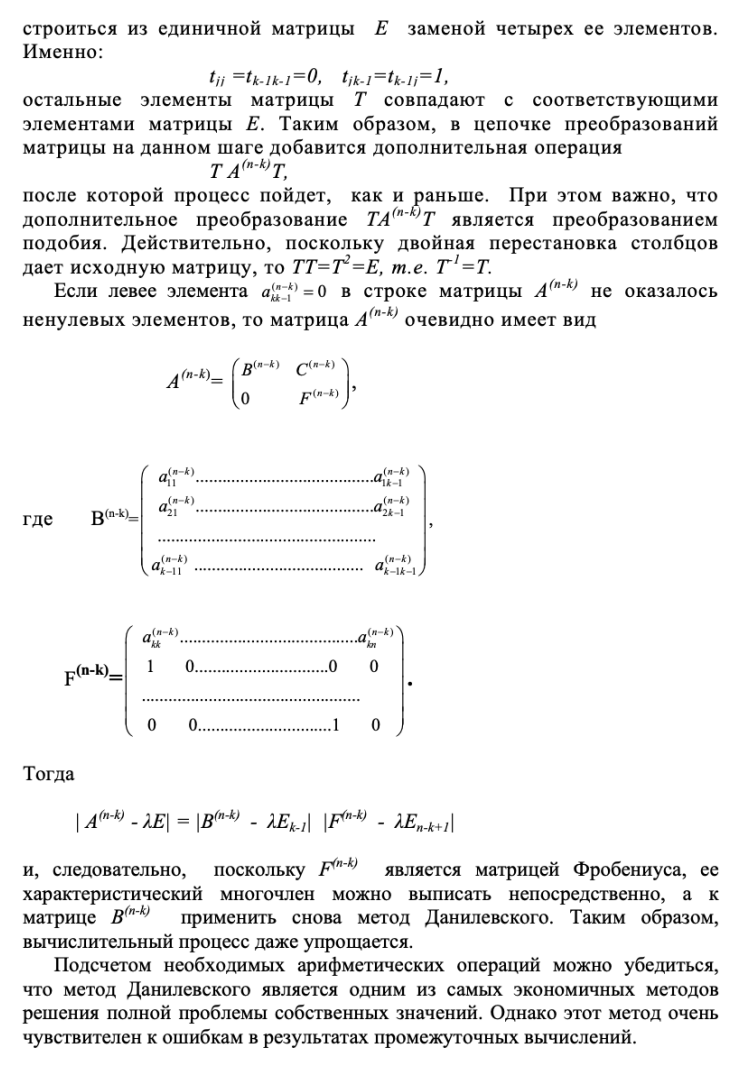
****

****

****

****

****

****

**Программная реализация.**

import numpy as np

def jacobi\_eigenvalue\_algorithm(A, tolerance): #Решение методом Якоби

if not check\_equal\_dim(A):

raise ValueError("Matrix isn't n, n dimension")

n = A.shape[0]

eigenvalues = np.diag(A)

eigenvectors = np.eye(n)

max\_iterations = 1000

iteration = 0

# Вычисление максимального недиагонального элемента

off\_diag = np.abs(A - np.diag(eigenvalues)).max()

while off\_diag > tolerance and iteration < max\_iterations:

# Находим индексы максимального недиагонального элемента

indices = np.argmax(np.abs(A - np.diag(eigenvalues)))

i = indices // n

j = indices % n

# Вычисляем угол поворота

if A[i, i] == A[j, j]:

theta = np.pi / 4

else:

theta = 0.5 \* np.arctan(2 \* A[i, j] / (A[i, i] - A[j, j]))

# Создаем матрицу вращения

rotation\_matrix = np.eye(n)

rotation\_matrix[i, i] = rotation\_matrix[j, j] = np.cos(theta)

rotation\_matrix[i, j] = -np.sin(theta)

rotation\_matrix[j, i] = np.sin(theta)

# Применяем матрицу вращения к матрице A

A = rotation\_matrix.T @ A @ rotation\_matrix

# Применяем матрицу вращения к матрице собственных векторов

eigenvectors = eigenvectors @ rotation\_matrix

# Обновляем собственные значения и проверяем условие остановки

eigenvalues = np.diag(A)

off\_diag = np.abs(A - np.diag(eigenvalues)).max()

iteration += 1

return eigenvalues, eigenvectors, iteration

def meth\_danil(matrix, tol=0.0001, verbose=0): #Решение методом Данилевского

# Проверяется, является ли матрица matrix квадратной

# (check\_equal\_dim - функция, которая проверяет равенство числа строк и столбцов матрицы).

if not check\_equal\_dim(matrix):

raise ValueError("Matrix isn't n, n dimension")

if verbose == 1:

print("Computing eigenvalues...")

a = matrix.copy()

f = matrix.copy()

n = a.shape[0]

s = np.eye(n)

for i in range(n - 1):

m = np.eye(n)

# Cоздается матрица m, которая является единичной, кроме строки n - 2 - i, которая заменяется на f[n - 1 - i].

m[n - 2 - i][:] = f[n - 1 - i][:]

f = np.dot(m, f) # умножаем A на M^(-1) слева

f = np.dot(f, np.linalg.inv(m)) # умножаем A на M справа

s = np.dot(s, np.linalg.inv(m)) # находим S

p = f[0] # выделяем p

p = p \* (-1)

p = np.insert(p, 0, 1)

eig\_val = np.roots(p)

eig\_vec = np.zeros(shape=(eig\_val.shape[0], n))

for j in range(0, eig\_val.shape[0]):

y = np.zeros(shape=(n, 1))

for i in range(0, n):

y[n-1-i] = eig\_val[j]\*\*i

x = np.dot(s, y) # находим собственный вектор

norm = np.linalg.norm(x)

for i in range(0, n):

eig\_vec[i][j] = x[i][0]/norm

return eig\_val, eig\_vec

def is\_matrix\_symmetric(matrix, eps) -> bool:

for i in range(1, len(matrix)):

for j in range(i):

if matrix[i][j] - matrix[j][i] >= eps:

return False

return True

def check\_equal\_dim(matrix):

if matrix.shape[0] != matrix.shape[1]:

return False

return True

C = np.array([[0.2, 0, 0.2, 0, 0],

[0, 0.2, 0, 0.2, 0],

[0.2, 0, 0.2, 0, 0.2],

[0, 0.2, 0, 0.2, 0],

[0, 0, 0.2, 0, 0.2]])

D = np.array([[2.33, 0.81, 0.67, 0.92, -0.53],

[0.81, 2.33, 0.81, 0.67, 0.92],

[0.67, 0.81, 2.33, 0.81, 0.92],

[0.92, 0.67, 0.81, 2.33, -0.53],

[-0.53, 0.92, 0.92, -0.53, 2.33]])

A = 6 \* C + D

print("Изначальная матрица: ")

print(A)

print()

eigenvalues, eigenvectors, iteration = jacobi\_eigenvalue\_algorithm(A, 0.0001)

print("Матрица собственных значений: ")

print(eigenvalues)

print()

print("Собственные векторы: ")

print(eigenvectors)

print()

print("Количество итераций: ", iteration)

print()

print()

eigenvalues, eigenvectors = meth\_danil(A)

print("Матрица собственных значений: ")

print(eigenvalues)

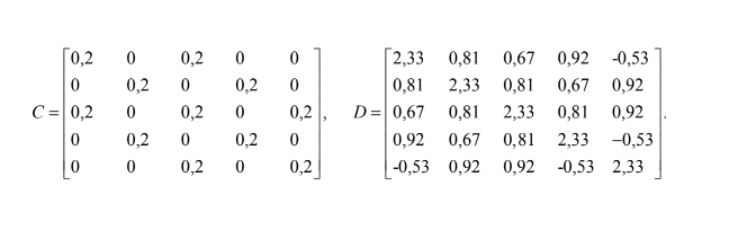
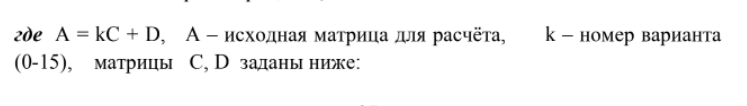
print()

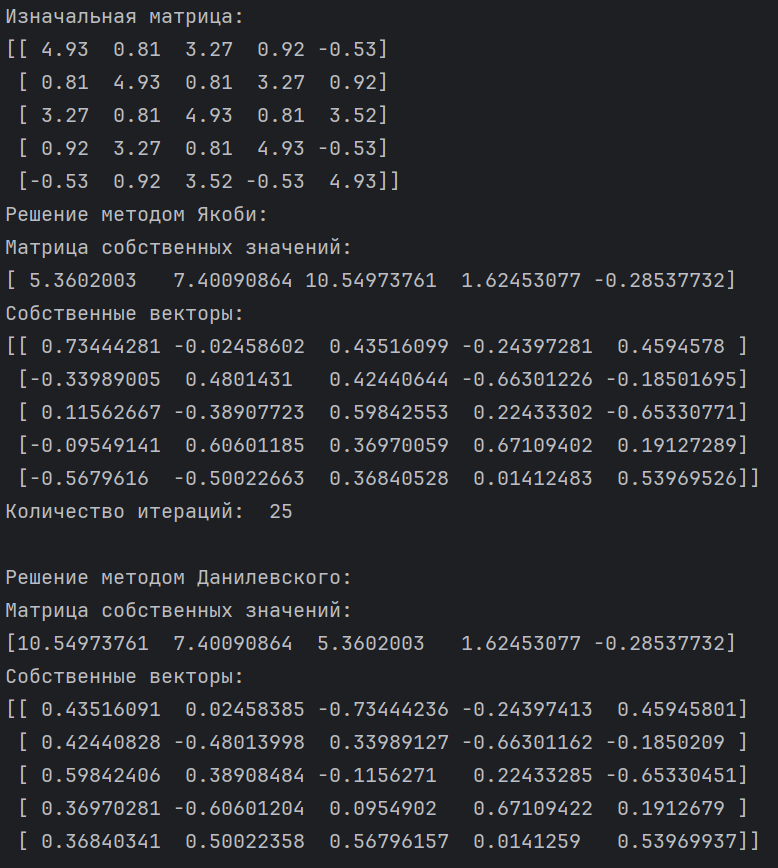
print("Собственные векторы: ")

print(eigenvectors)

**Решение задания:**

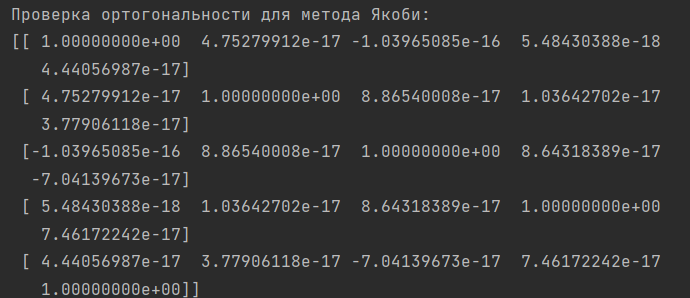
С точностью 0.0001 вычислить собственные значения и собственные

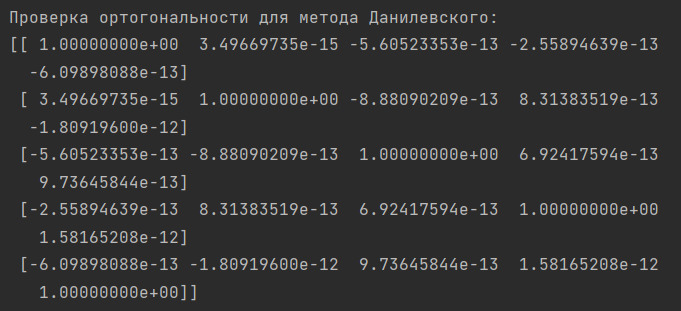
вектора матрицы A.**** ****



**Оценка точности**

Т.к скалярное произведение собственных векторов матрицы рано 0, оценим его с точностью 0,0001.Проверку точности будем так же проводить при помощи пакета numpy:

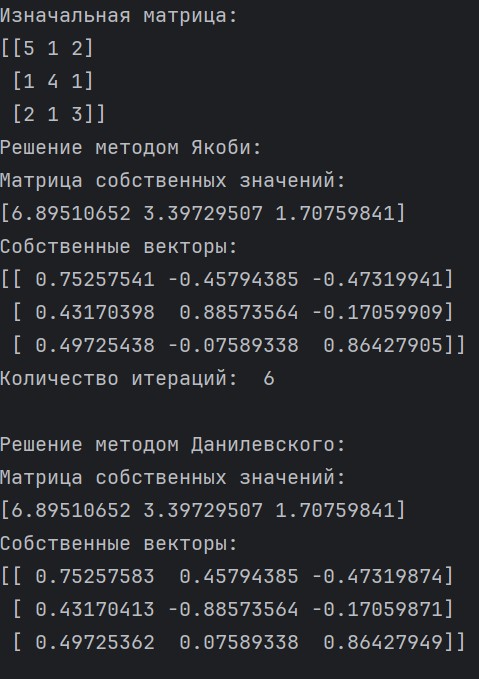




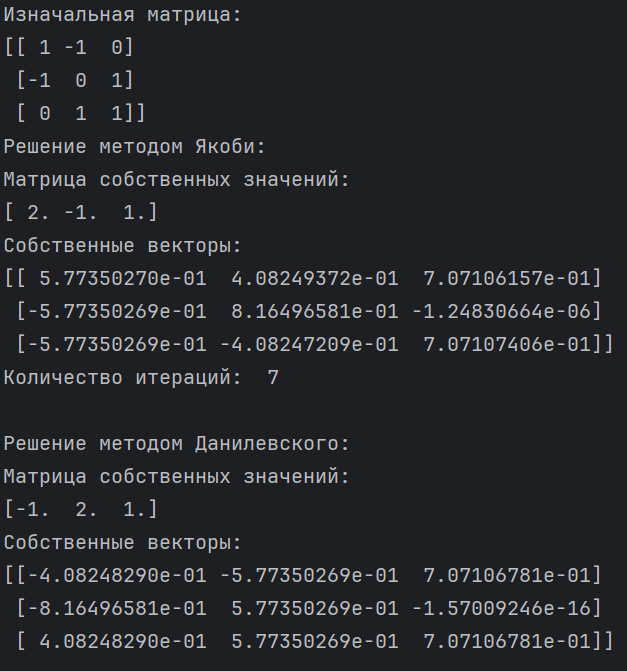
Как видно, точность 0,0001 достичь удалось. Кроме того, для метода Якоби удалось достичь точности 10-16, а для метода Данилевского 10-12.

**Тестовые примеры**

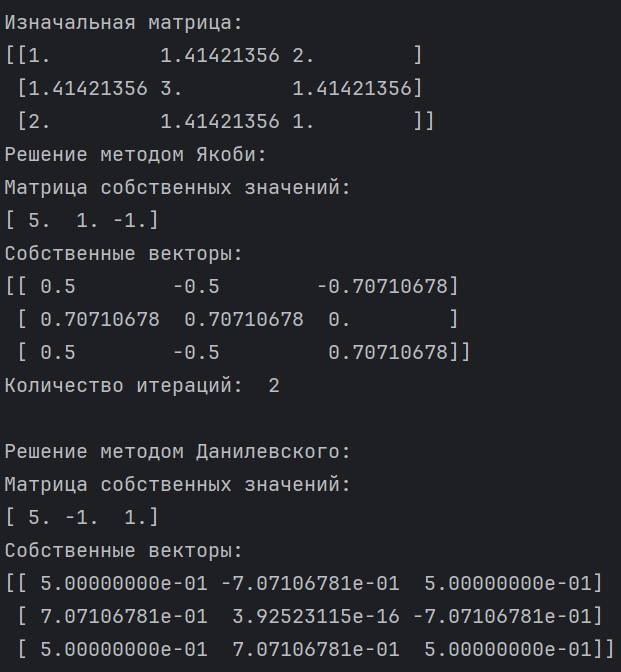
**Тестовый пример 1.**

****

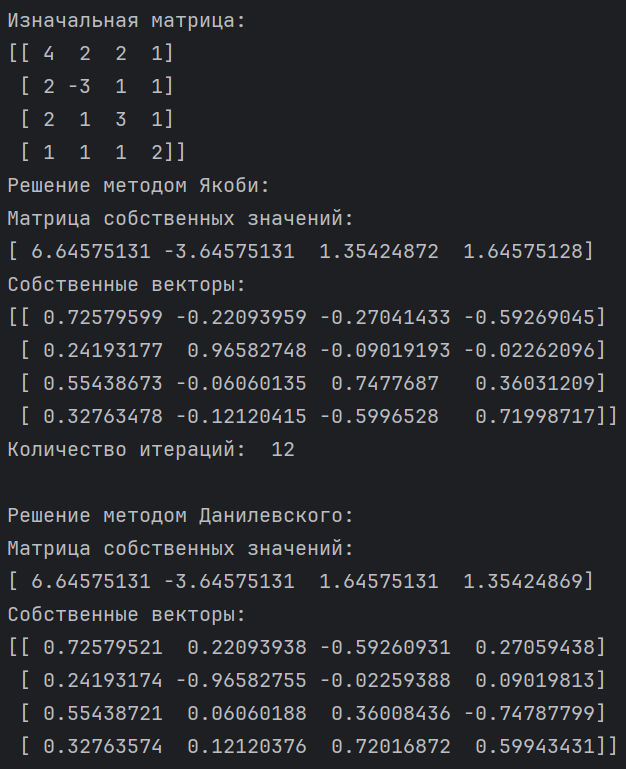
**Тестовый пример 2.**

****

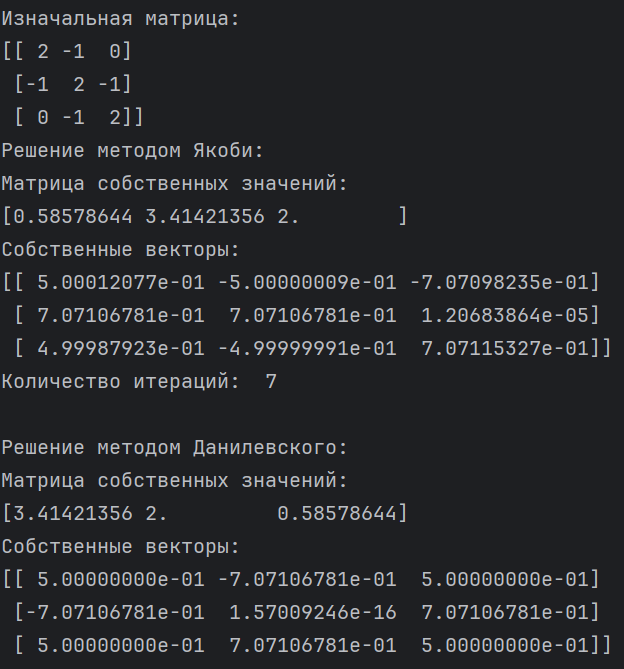
**Тестовый пример 3.**

****

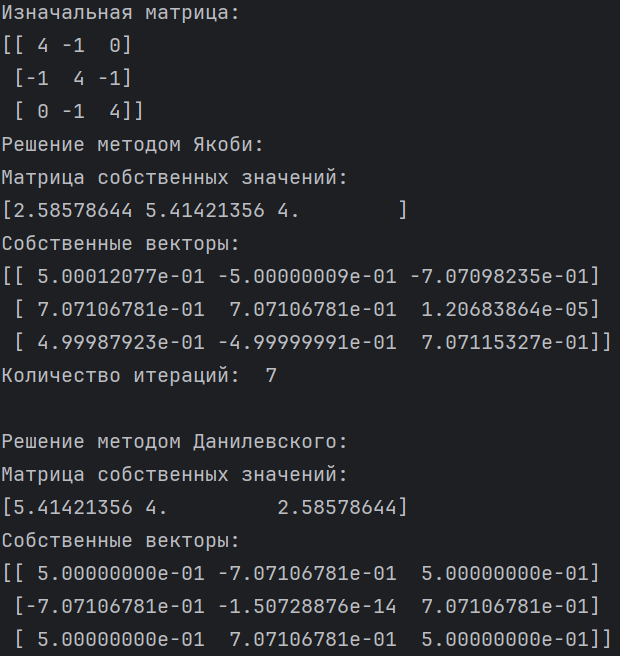
**Тестовый пример 4.**

****

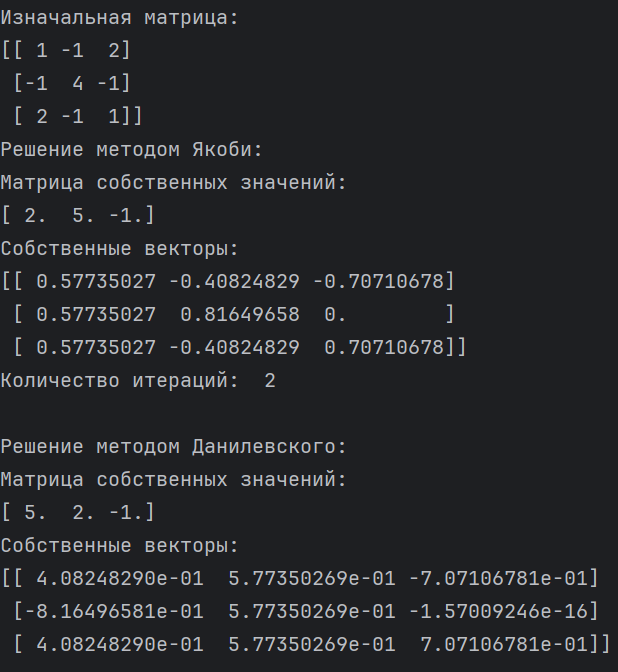
**Тестовый пример 5.**

****

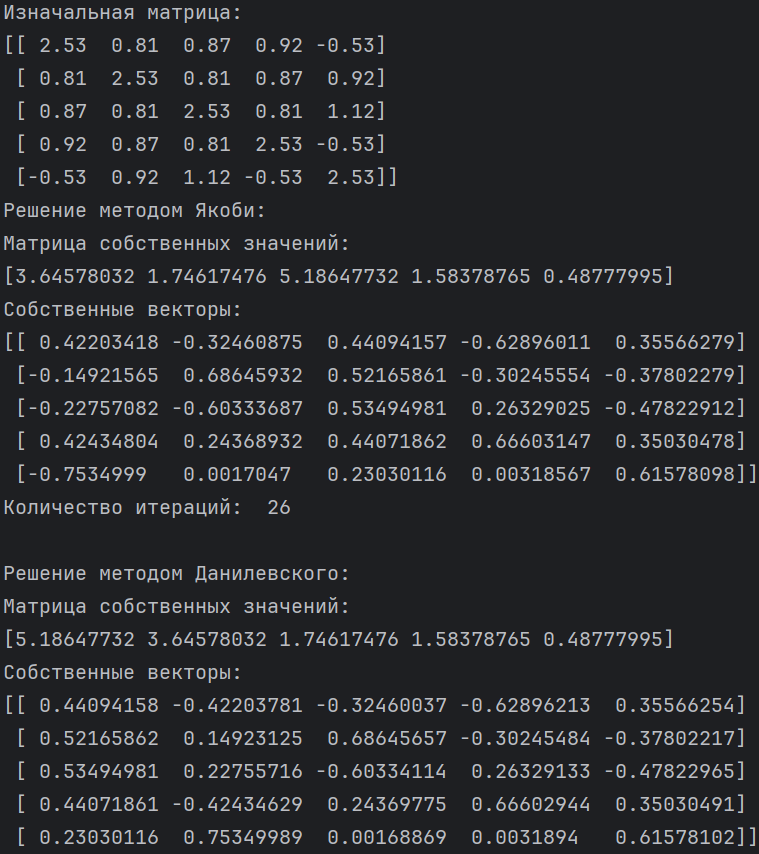
**Тестовый пример 6.**

****

**Тестовый пример 7.**

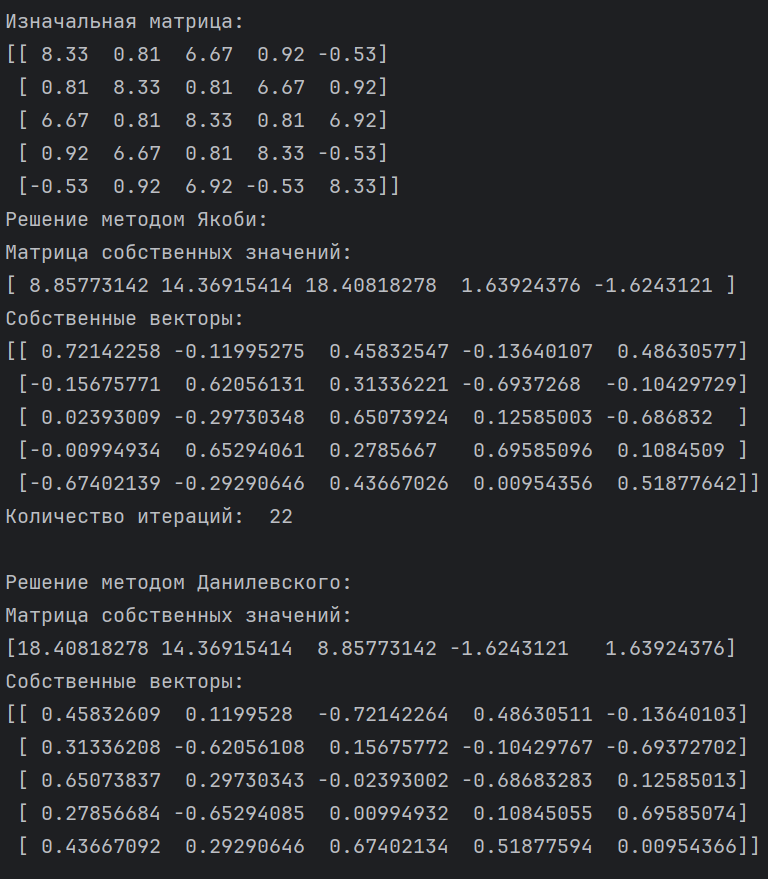
****

**Тестовый пример 8.**

При k = 1(Матрицы из задания): 

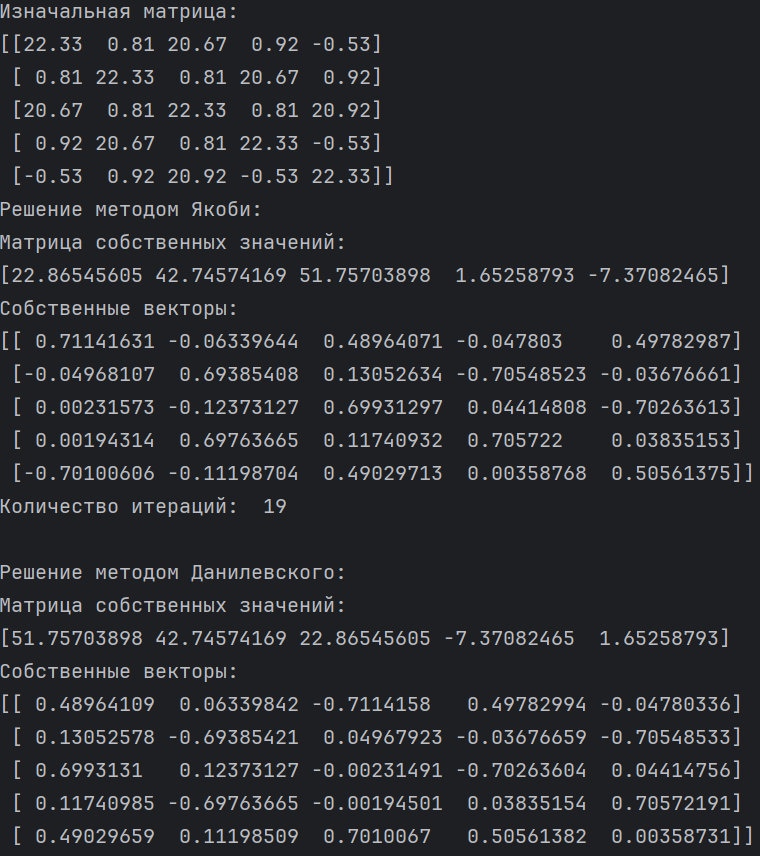
**Тестовый пример 9.**

При k = 30(Матрицы из задания):

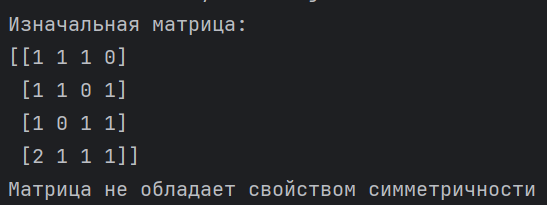
****

**Тестовый пример 10.**

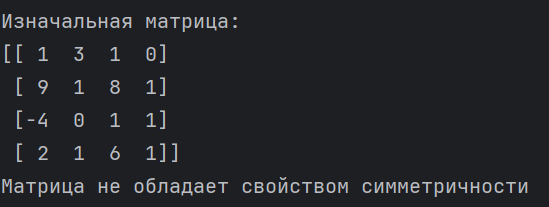
При k = 100(Матрицы из задания):



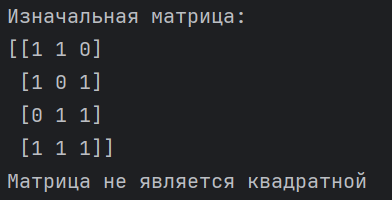
**Тестовый пример 11.**



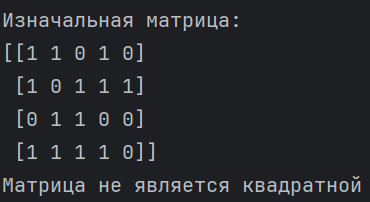
**Тестовый пример 12.**

****

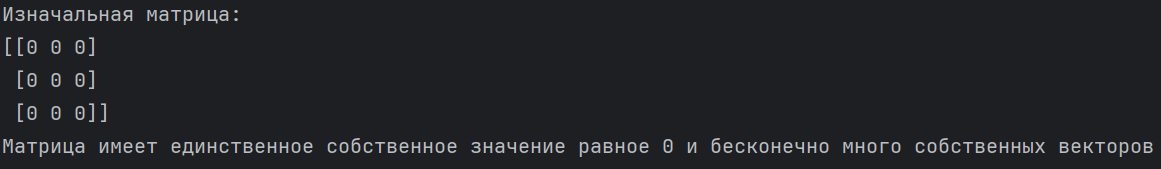
**Тестовый пример 13.**

****

**Тестовый пример 14.**

****

**Тестовый пример 15.**

****

# **Выводы**

В ходе проделанной работы, были рассмотрены матрицы вращения в n мерных пространствах, а также их применение в различных алгоритмах. Одним из них является метод вращения Якоби, который позволяет вычислять все собственные значения для симметричных матриц.

Разобраны 15 тестовых примеров, которые говорят о том, что алгоритм корректно отрабатывает для любой симметричной матрицы.