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 253505

Сенько Никита Святославович

Проверил: Анисимов Владимир Яковлевич

Минск 2023

**Содержание**

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

Вариант 10

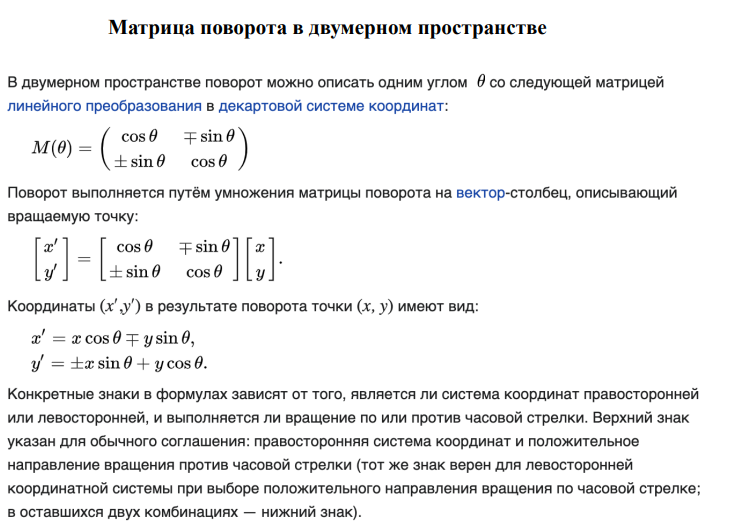
# **Цели выполнения задания**

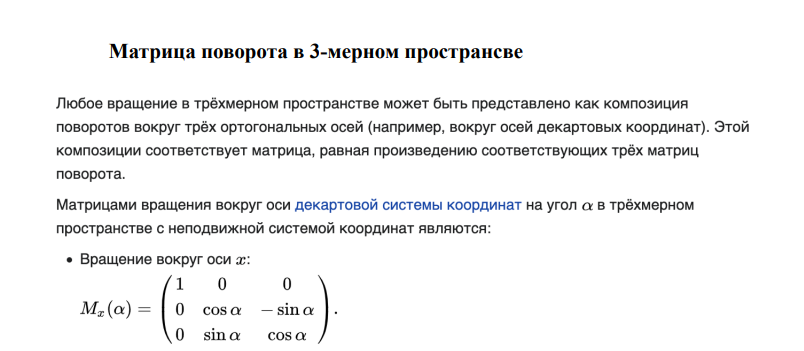
1) Рассмотреть методы вычисления собственных значений и векторов

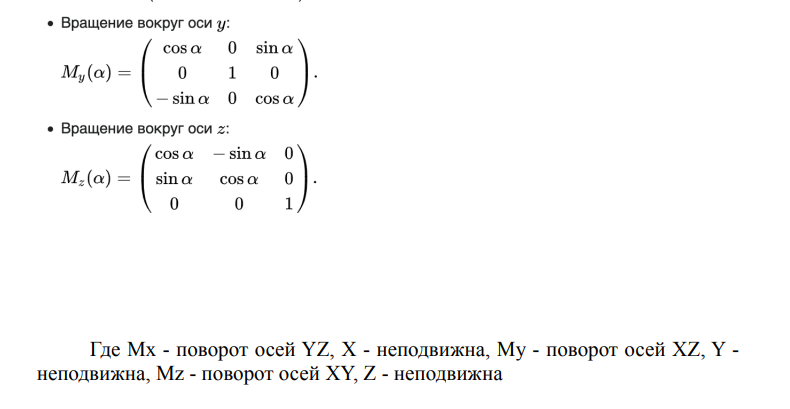
2) Реализовать методы вращений Якоби и метод Данилевского

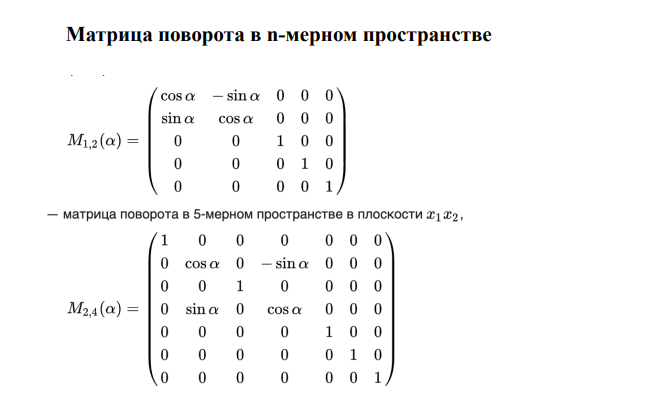
# **Краткие теоретические сведения**

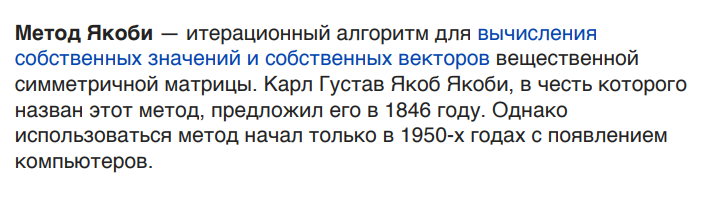
Итеративные алгоритмы решают задачу вычисления собственных значений путём построения последовательностей, сходящихся к собственным значениям. Некоторые алгоритмы дают также последовательности векторов, сходящихся к собственным векторам. Чаще всего последовательности собственных значений выражаются через 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.

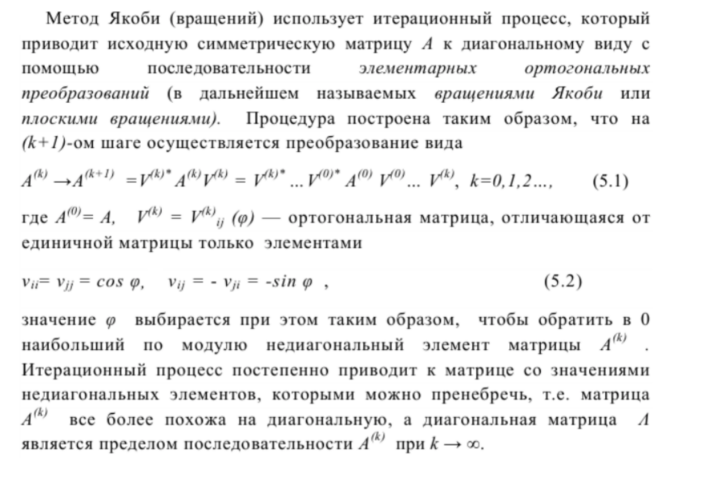


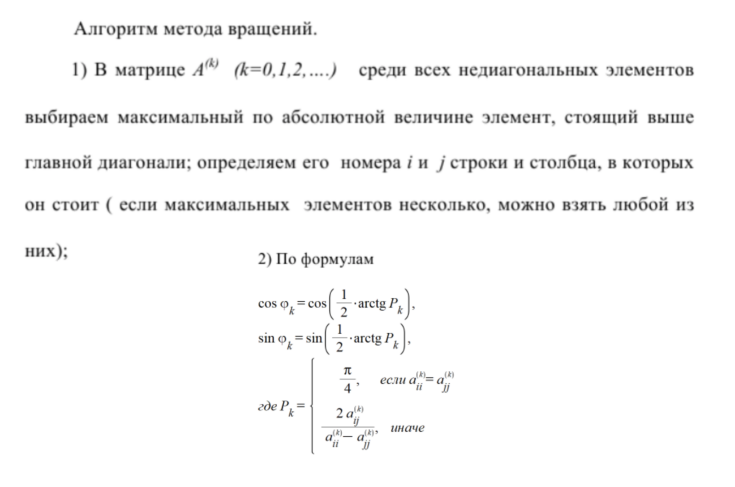


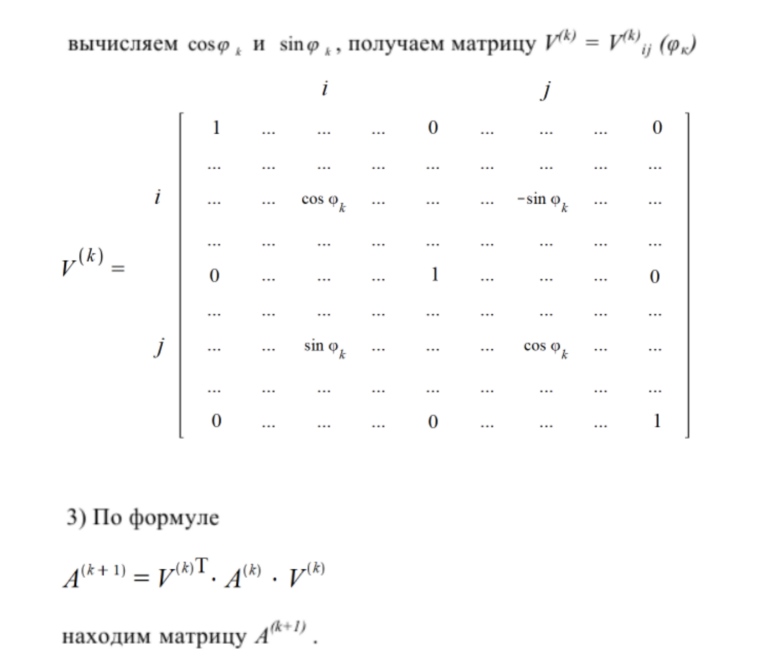


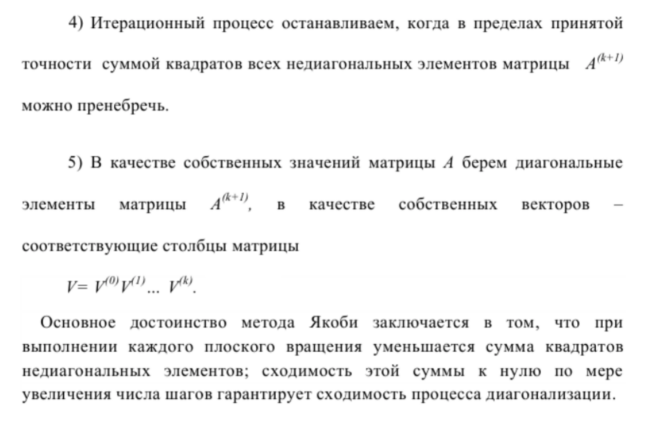


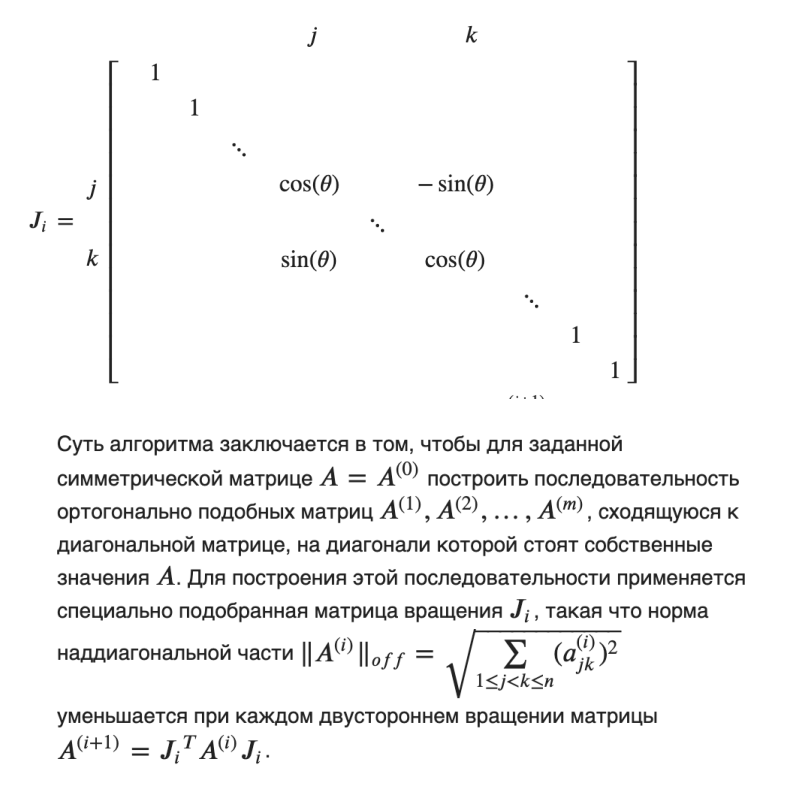


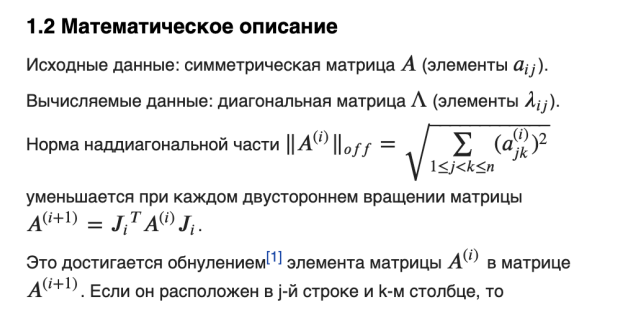


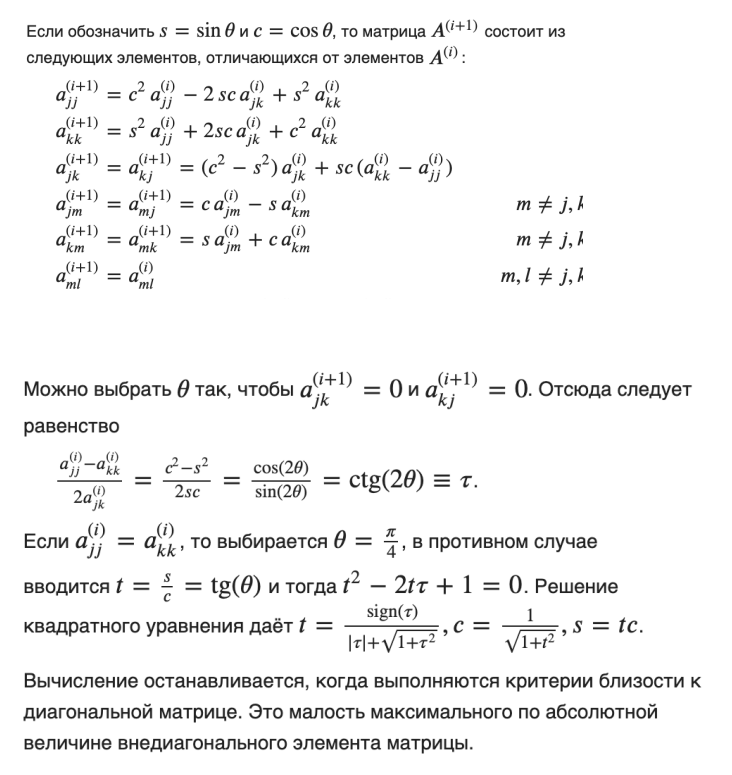


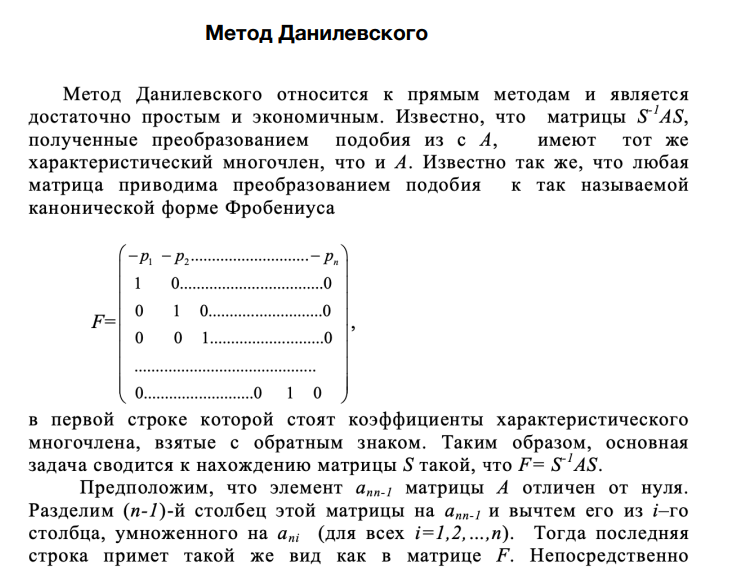
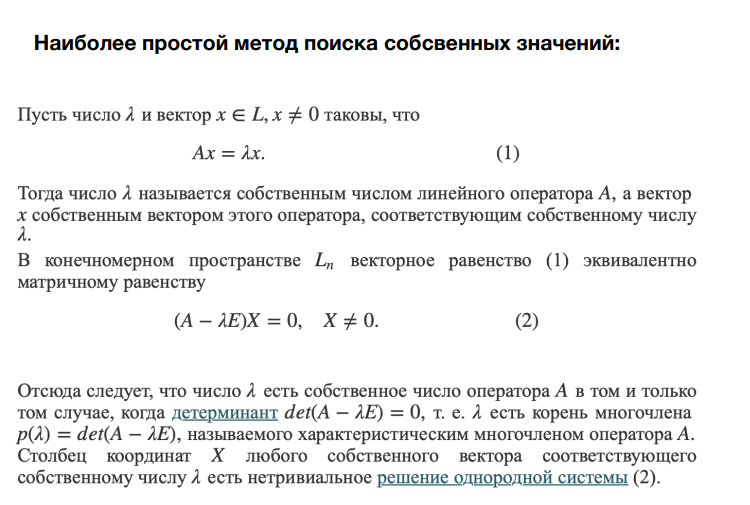


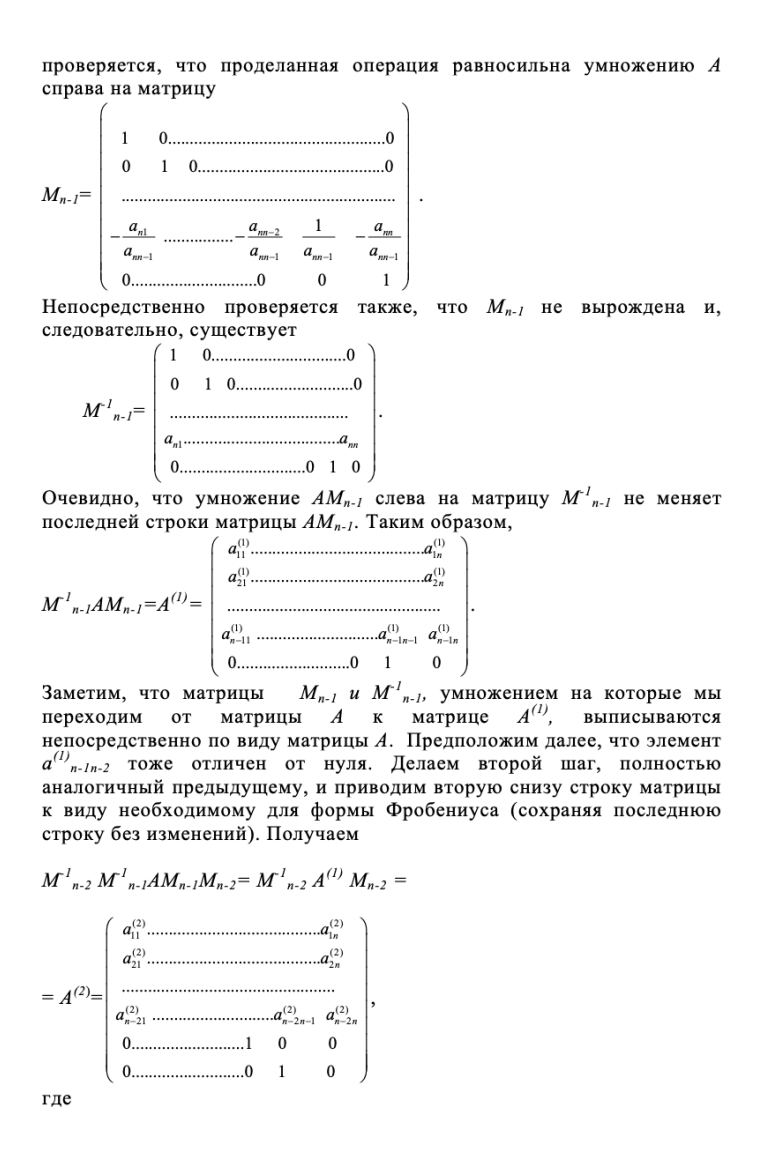


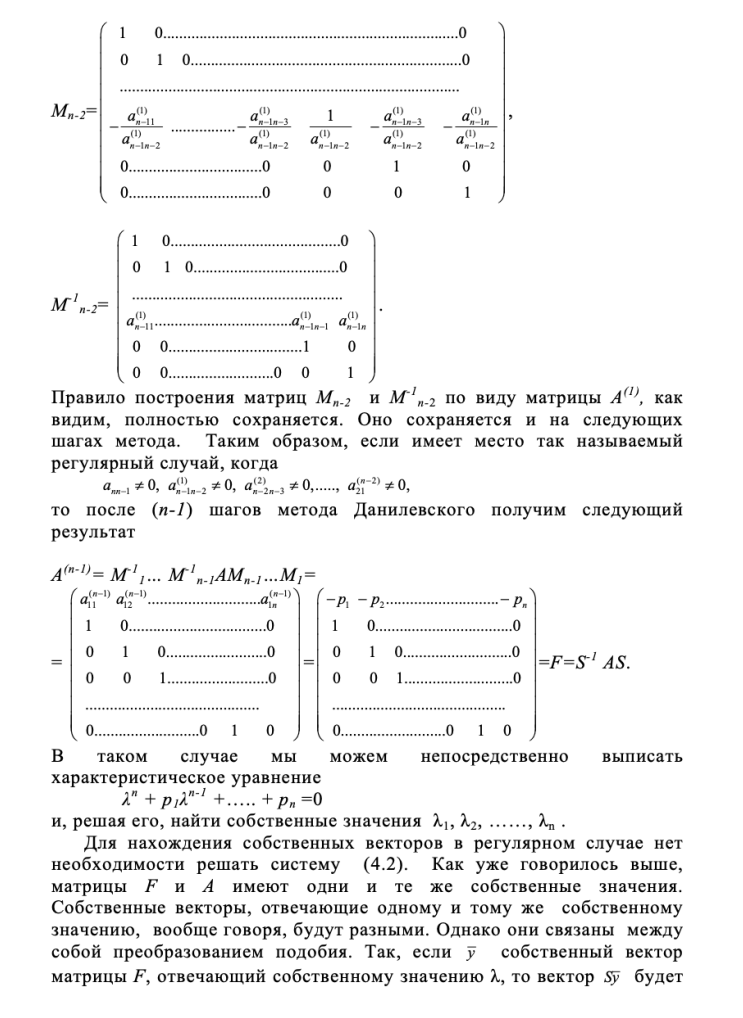


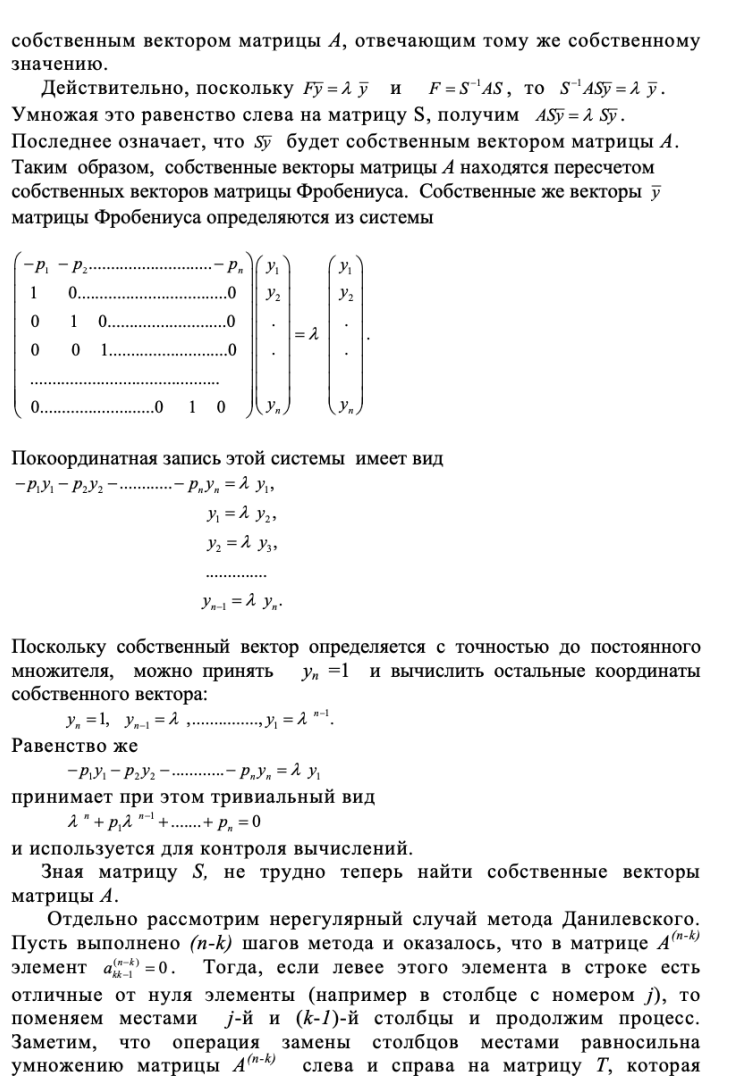


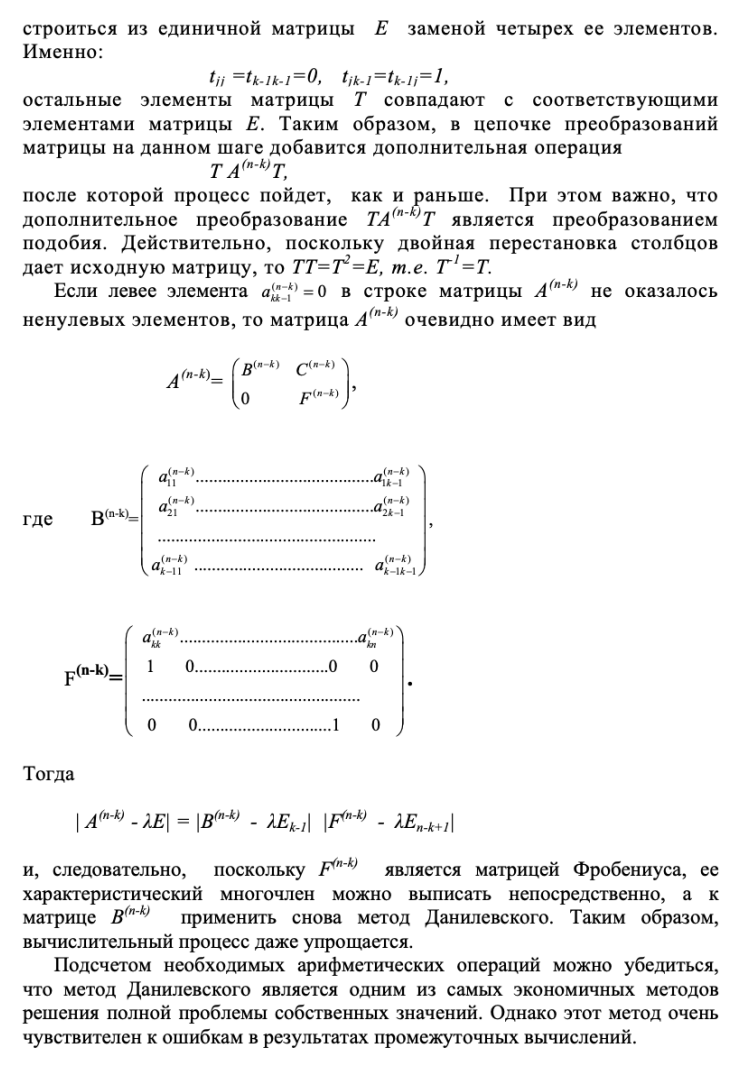












**Программная реализация.**

**Решение методом Якоби:**

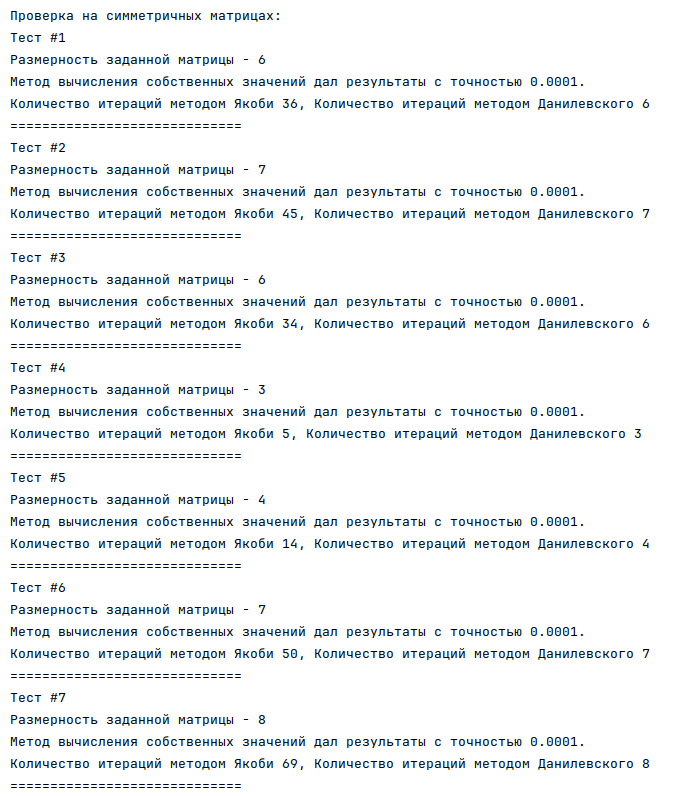
def jacobi\_eigenvalue\_algorithm(A, tolerance):  
 if not check\_equal\_dim(A):  
 raise ValueError("Matrix isn't n, n dimension")  
 n = A.shape[0]  
 eigenvalues = np.diag(A)  
 eigenvectors = np.eye(n)  
  
 max\_iterations = 1000  
 iteration = 0  
 off\_diag = np.abs(A - np.diag(eigenvalues)).max()  
  
 while off\_diag > tolerance and iteration < max\_iterations:  
 indices = np.argmax(np.abs(A - np.diag(eigenvalues)))  
 i = indices // n  
 j = indices % n  
  
 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]))  
  
 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)  
  
 A = rotation\_matrix.T @ A @ rotation\_matrix  
 eigenvectors = eigenvectors @ rotation\_matrix  
  
 eigenvalues = np.diag(A)  
 off\_diag = np.abs(A - np.diag(eigenvalues)).max()  
  
 iteration += 1  
  
 return eigenvalues, eigenvectors, iteration

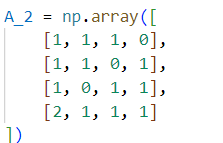
**Решение методом Данилевского:**

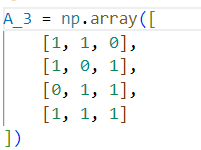
def meth\_danil(matrix, tol=0.0001, verbose=0):  
 if not check\_equal\_dim(matrix):  
 raise ValueError("Matrix isn't n, n dimension")  
 if verbose == 1:  
 print("Computing eigenvalues...")  
  
 a = matrix.copy()  
 f = matrix.copy()  
 n = a.shape[0]  
 s = np.eye(n)  
  
 for i in range(n - 1):  
 m = np.eye(n)  
 m[n - 2 - i][:] = f[n - 1 - i][:]  
 f = np.dot(m, f)  
 f = np.dot(f, np.linalg.inv(m))  
 s = np.dot(s, np.linalg.inv(m))  
  
 p = f[0]  
 p = p \* (-1)  
 p = np.insert(p, 0, 1)  
 eig\_val = np.roots(p)  
 eig\_vec = np.zeros(shape=(eig\_val.shape[0], n))  
  
 iteration = 0  
  
 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)  
 norm = np.linalg.norm(x)  
 for i in range(0, n):  
 eig\_vec[i][j] = x[i] / norm  
 iteration += 1  
  
 return eig\_val, eig\_vec, iteration

# **Система автоматического тестирования**

def test\_methods(num\_tests: int = 10, issym=True):  
 np.random.seed(42)  
 tolerance = 0.0001  
  
 for i in range(num\_tests):  
 n = np.random.randint(3, 9)  
 if issym:  
 a = np.random.rand(n, n)  
 a = 0.5 \* (a + a.T) *# Создание симметричной матрицы* else:  
 a = np.random.rand(n, n)  
 np.fill\_diagonal(a, np.diagonal(a) + n)  
 print(f"Тест #{i + 1}")  
 expected\_solution = np.linalg.eig(a).eigenvalues  
 val = jacobi\_eigenvalue\_algorithm(a, tolerance)  
 val1 = meth\_danil(a, tolerance)  
 if val and val1:  
 eig\_val, eig\_vec, it1 = val  
 eig\_val1, eig\_vec1, it2 = val1  
 else:  
 continue  
 eigen\_correct = True  
 if eig\_val1 is None or eig\_val is None:  
 continuesorted\_indices = np.argsort(eig\_val)  
 eig\_val = eig\_val[sorted\_indices]  
 sorted\_indices1 = np.argsort(eig\_val1)  
 eig\_val1 = eig\_val1[sorted\_indices1]  
 sorted\_indices2 = np.argsort(expected\_solution)  
 expected\_solution = expected\_solution[sorted\_indices2]for j in range(len(expected\_solution)):  
 if abs(abs(eig\_val[j]) - abs(expected\_solution[j])) > tolerance:  
 eigen\_correct = False  
 for j in range(len(expected\_solution)):  
 if abs(abs(eig\_val1[j]) - abs(expected\_solution[j])) > tolerance:  
 eigen\_correct = False  
  
 print(f"Размерность заданной матрицы - {n}")  
 if eigen\_correct:  
 print("Метод вычисления собственных значений дал результаты с точностью 0.0001.")  
 print(f"Количество итераций методом Якоби {it1}, Количество итераций методом Данилевского {it2}")  
 else:  
 print("Метод вычисления собственных значений дал результаты с недопустимой точностью.")  
  
 print("=============================")







****

**Решение задания:**

**Вариант 10.**

С точностью 0.0001 вычислить собственные значения и собственные

вектора матрицы A.

# 

**Выводы**

В ходе проделанной работы были рассмотрены матрицы вращения в n-мерных пространствах и их применение в различных алгоритмах. Одним из таких алгоритмов является метод вращения Якоби, который эффективно вычисляет все собственные значения для симметричных матриц.

В работе были представлены три тестовых примера, демонстрирующих правильную работу алгоритма для любой симметричной матрицы. Кроме того, была проведена сравнительная проверка результатов с использованием метода из библиотеки NumPy для определения собственных векторов и значений.

Важно отметить, что в ходе исследования было обнаружено, что метод Данилевского работает еще более быстро и эффективно для вычисления собственных значений симметричных матриц.

**Список использованной литературы**

* + - 1. Минченко Л.И. Краткий курс численного анализа. Учебное пособие по курсу «Методы численного анализа» – Мн.: БГУИР, 2006. – 92 с.
      2. Савчук, В.Ф. Методы численного анализа : электрон. курс лекций – Брест : электрон. издание БрГУ, 2013. – 403 с.
      3. Зинина А. И., Копнина В. И. Численные методы линейной и нелинейной алгебры – Саратов, 2016 – 152 с.
      4. Зенков, А.В. Численные методы : учеб. Пособие — Екатеринбург , 2016.— 124 с.