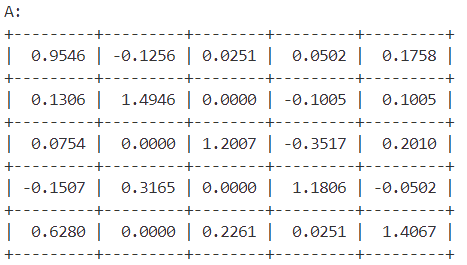
**Постановка задачи**

Построить собственный многочлен матрицы , найти минимальное собственное значение и собственный вектор, соответствующий этому собственному значению, где:



Заданная точность: eps=1e-5.

**Необходимо:**

1. Построить: – собственный многочлен матрицы , найдя коэффициенты используя метод Данилевского.
2. Проверить точность вычисления вычислив:

;

.

1. Реализовать степенной метод через скалярное произведение;
2. Найти минимальное собственное значение ;
3. Найти собственный вектор x, соответствующий найденному собственному значению;
4. Оценить точность:

;

;

**Алгоритм решения**

Работаем с матрицей

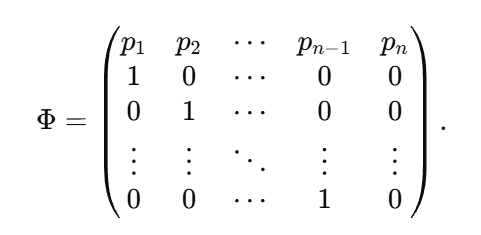
1. **Алгоритм метода Данилевского**

Метод Данилевского относится к прямым методам решения проблемы

собственных значений. Метод основан на подобном преобразовании матрицы: преобразованиями матрица приводится к канонической форме Фробениуса, которая содержит коэффициенты характеристического многочлена.

Матрица приводится к Ф, в результате последовательного домножения справа на и слева на , k = 1, ..., n -1.

Таким образом справедлива формула: , где , положим =. В итоге получим = Ф.



где pi - соответствующий коэффициент (с противоположным знаком) собственного многочлена матрицы , i=.

Для матрицы справедливы следующие формулы:

*, , ,* где

Для матрицы имеем:

1. **Алгоритм степенного метода**

Для того, чтобы найти для матрицы , сведём задачу к нахождению для матрицы .

Теперь перед нами стоит задача нахождения наибольшего по модулю собственного значения и соответствующего ему собственного вектора.

Обозначим собственные значения следующим образом: λ1, . . ., λ𝑛. Будем считать, что все собственные значения матрицы A перенумерованы в порядке невозрастания модулей:

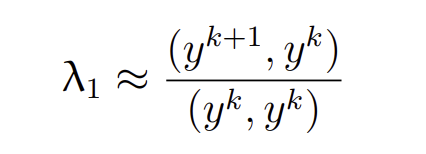
|λ1| > |λ2| ≥ … ≥ |λn|.

Пусть – произвольный ненулевой вектор (положим =[1, 0, …, 0]).

Перейдём к итерационному процессу:

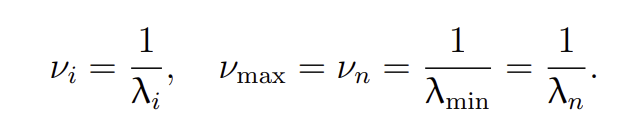
, k = 0, 1, 2, …

В качестве λ1 можно взять:



В качестве критерия для остановки итерационного процесса берём:

Обозначим через , 𝑖 = собственные значения матрицы . Далее, зная соотношения между матрицей и обратной к ней, получаем



Таким образом минимальное собственное значение будет найдено по формуле: .

Собственный вектор x = .

**Листинг**

import numpy as np

from tabulate import tabulate

from sympy import Symbol, solve

A = np.loadtxt(fname="A.txt", dtype=float)

print(f"A:\n{tabulate(A, tablefmt='grid', floatfmt='.4f')}")

A = A.T @ A

print(f"A.T @ A:\n {tabulate(A, tablefmt='grid', floatfmt='.4f')}")

def format\_print(X, p, r1, r2, eigenvalue, eigenvector, r\_eigenvalue, r\_eigenvector, c=None):

    p \*= -1

    x = [f"x^{i}" if i != 1 else "x" for i in range(5, 0, -1)]

    polynom = "p(x)="

    for i, j in zip(x, p):

        polynom += i

        polynom += f" - {-round(j, 3)}" if np.sign(j) == -1 else f" + {round(j, 3)}"

    print(f'1)Frobenius normal form:\n{tabulate(X, tablefmt="grid", floatfmt=".3f")}\n',\

        f'2)Characteristic polynomial:\n{polynom}\n',\

        f'3)r1 = p1 - SpA = {r1:.3e}\n',\

        f'4)r2 = p5 - detA = {r2:.3e}\n',\

        f'5)min eigenvalue: {eigenvalue}\n',\

        f'6)eigenvector:\n{tabulate([eigenvector], tablefmt="grid", floatfmt=".5f")}\n',\

        f'7)r of eigenvalue: {r\_eigenvalue}\n',\

        f'8)r of eigenvector:\n{tabulate([r\_eigenvector], tablefmt="grid", floatfmt=".3e")}\n',\

          "" if c is None else f'9)count of iterations: {c}')

def format\_print\_danilevsky(eigenvalue, eigenvector, r\_eigenvalue, r\_eigenvector, c=None):

    print(

        '\nDanilevsky eigenvector:\n',\

        f'5)min eigenvalue: {eigenvalue}\n',\

        f'6)eigenvector:\n{tabulate([eigenvector], tablefmt="grid", floatfmt=".5f")}\n',\

        f'7)r of eigenvalue: {r\_eigenvalue}\n',\

        f'8)r of eigenvector:\n{tabulate([r\_eigenvector], tablefmt="grid", floatfmt=".3e")}\n',

          "" if c is None else f'9)count of iterations: {c}'

    )

def danilevsky\_method(A: np.ndarray):

    X = A.copy()

    n = X.shape[0]

    s = np.eye(n)

    n -= 1

    for i in np.arange(n):

        ones\_left = np.eye(n+1)

        ones\_left[n-1-i] = X[n-i]

        ones\_right = ones\_left.copy()

        ones\_right[n-1-i] /= -X[n-i, n-1-i]

        ones\_right[n-1-i, n-1-i] = 1 / X[n-i, n-1-i]

        X = ones\_left @ X @ ones\_right

        s = s @ ones\_right

    p = X[0]

    r1 = p[0] - np.trace(A)

    detA = np.linalg.det(A)

    r2 = p[n] - detA

    return X, r1, r2, s, p

#find min eigenvalue

def power\_iteration(A: np.ndarray, num\_iter: int=1000, tol: float=1e-5):

    n = A.shape[0]

    u = np.zeros(n)

    u[0] = 1

    prev\_eigenvalue = 0

    c = 0

    u\_prev = u

    for k in range(1, num\_iter + 1):

        c += 1

        v = A @ u

        v\_norm = np.linalg.norm(v, np.inf)

        u = v / v\_norm

        eigenvalue = (v @ u\_prev) / (u\_prev @ u\_prev)

        if abs(eigenvalue - prev\_eigenvalue) < tol:

            break

        prev\_eigenvalue = eigenvalue

        u\_prev = u

    return eigenvalue, u, c

def danilevsky\_power\_method(A: np.ndarray, type\_eigenvector='danilevsky'):

    X = np.linalg.inv(A)

    n = X.shape[0]

    X, r1, r2, s, p = danilevsky\_method(X)

    eigenvalue, u, c = power\_iteration(X)

    r\_eigenvalue = np.sum([p[n-1-i] \* (eigenvalue \*\* i) for i in range(n)]) - eigenvalue \*\* (n)

    if type\_eigenvector == 'danilevsky':

        y = np.array([eigenvalue \*\* i for i in np.arange(n-1, -1, -1)])

        eigenvector = s @ y

        r\_eigenvector = X @ eigenvector - eigenvalue \* eigenvector

    else:

        r\_eigenvector = X @ u - eigenvalue \* u

        eigenvector = u

    return X, p, r1, r2, c, 1/eigenvalue, eigenvector, r\_eigenvalue, r\_eigenvector

X, p, r1, r2, c, eigenvalue, eigenvector, r\_eigenvalue, r\_eigenvector = danilevsky\_power\_method(A)

format\_print(X, p, r1, r2, eigenvalue, eigenvector, r\_eigenvalue, r\_eigenvector, c)

\_, \_, \_, \_, c, eigenvalue, eigenvector, r\_eigenvalue, r\_eigenvector = danilevsky\_power\_method(A, type\_eigenvector='')

format\_print\_danilevsky(eigenvalue, eigenvector, r\_eigenvalue, r\_eigenvector, c)

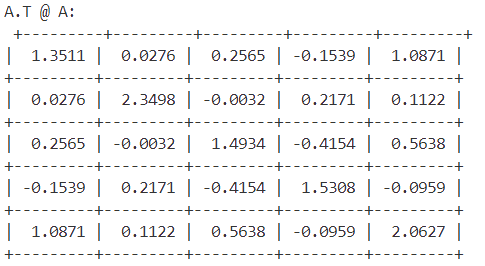
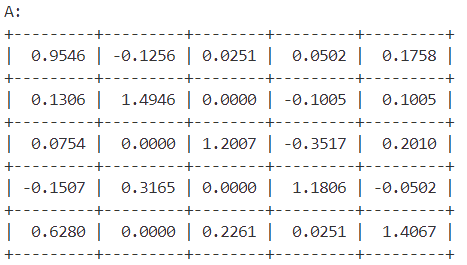
eigs = np.sort(np.linalg.eig(A).eigenvalues)

print(eigs)

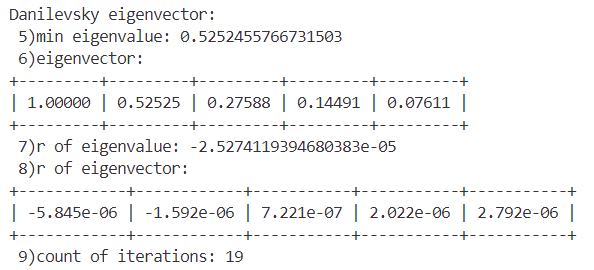
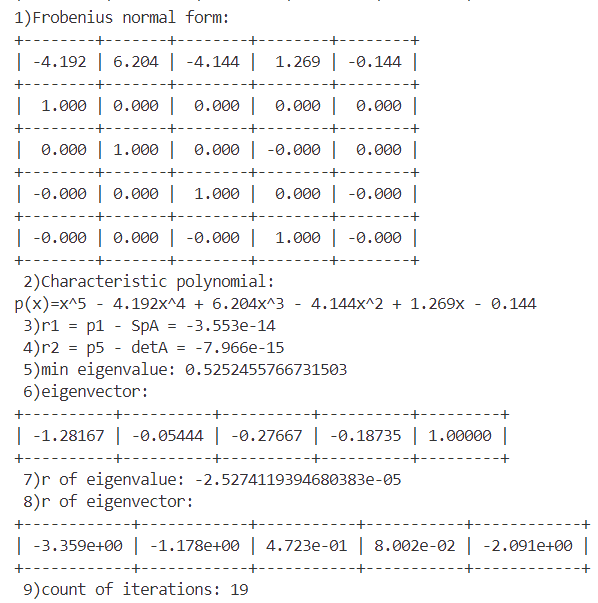
print(abs(eigs[-2])/abs(eigs[-1]))

**Результаты и их анализ**

Матрица входные данные:



Результаты:



Для сравнения вычислим собственные значения при помощи стандартной библиотеки:



Метод Данилевского является точным методом, это подтверждает маленькая погрешность. , , отсюда => выполняется достаточное условие сходимости степенного метода.На основе результатов вычислений можно сделать следующие выводы. Значения r1 и r2, которые близки к нулю, подтверждают корректность найденных коэффициентов собственного многочлена матрицы А. Значение r3​ (r of eigenvalue) свидетельствует о высокой точности вычислений собственных значений и векторов. Норма вектора r4 (r of eigenvector), это может быть связано с накоплением погрешностей в результате вычислений.