Постановка задачи

Построить собственный многочлен матрицы $\tilde{A} = A^T A$, найти минимальное собственное значение $\boldsymbol{\lambda}_{min}(\tilde{A})$ и собственный вектор, соответствующий этому собственному значению, где:

A:

0.9546	-0.1256	0.0251	0.0502	0.1758
0.1306	1.4946	0.0000	-0.1005	0.1005
0.0754	0.0000	1.2007	-0.3517	0.2010
-0.1507	0.3165	0.0000	1.1806	-0.0502
0.6280	0.0000	0.2261	0.0251	1.4067

Заданная точность: eps=1e-5.

Необходимо:

1. Построить: $\mathrm{P_n}(\lambda) = (\lambda^{\mathrm{n}} - \mathrm{p}_1\lambda^{\mathrm{n}-1} - \ldots - \mathrm{p}_{n-1}\lambda - \mathrm{p_n})$ – собственный многочлен матрицы $\tilde{A}$, найдя коэффициенты используя метод Данилевского.
2. Проверить точность вычисления вычислив:
   $r_1 = p_1 - sp\tilde{A}$;
   $r_2 = p_n - det\tilde{A}$.
3. Реализовать степенной метод через скалярное произведение;
4. Найти минимальное собственное значение $\boldsymbol{\lambda}_{min}(\tilde{A})$;
5. Найти собственный вектор x, соответствующий найденному собственному значению;
6. Оценить точность:
   $r_3 = P_n(\boldsymbol{\lambda}_{\mathrm{min}})$;
   $r_4 = \tilde{A}x - \boldsymbol{\lambda}_{min}x$;

Алгоритм решения

Работаем с матрицей $\tilde{A} = A^T A$

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 $\tilde{A}$ приводится к Ф, в результате последовательного домножения справа на M_{n-k} и слева на M_{n-k}^{-1}, k = 1, ..., n -1.

Таким образом справедлива формула: $\tilde{A}_k = M_{n-k}^{-1} A_{k-1} M_{n-k}$, где $k = \overline{1, n-1}$, положим $\tilde{A}_0$=$\tilde{A}$. В итоге получим $\tilde{A}_{n-1}$= Ф.

$$\Phi = \begin{pmatrix} p_1 & p_2 & \cdots & p_{n-1} & p_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

где p_i - соответствующий коэффициент (с противоположным знаком) собственного многочлена матрицы $\tilde{A}$, i=$\overline{1, n}$.

Для матрицы M_{n-k} справедливы следующие формулы:

$m_{n-k,j} = -\frac{a_{n-k-1,j}}{a_{n-k-1,n-k}}, j \neq n-k;$ $\qquad m_{n-k,j} = \frac{1}{a_{n-k-1,n-k}}, j = n-k,$ где $j = \overline{1, n}, k = \overline{1, n-1}$

Для матрицы M_{n-k}^{-1} имеем:

$$m_{n-k,j}^{-1} = a_{n-k-1,j}, \qquad j = \overline{1, n}, k = \overline{1, n-1}$$

2. Алгоритм степенного метода

Для того, чтобы найти $\boldsymbol{\lambda}_{min}$ для матрицы $\tilde{A}$, сведём задачу к нахождению $\boldsymbol{\lambda}_{max}$ для матрицы $\tilde{A}^{-1}$.

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 А перенумерованы в порядке невозрастания модулей:

$|\lambda 1| > |\lambda 2| \geq \ldots \geq |\lambda n|$.

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

Перейдём к итерационному процессу:

$$y^{k+1} = Ay^k,\ k = 0, 1, 2, \ldots$$

В качестве λ1 можно взять:

$$\lambda_1 \approx \frac{(y^{k+1}, y^k)}{(y^k, y^k)}$$

В качестве критерия для остановки итерационного процесса берём:

$$\left||\lambda_1^{k+1}| - |\lambda_1^k|\right| \leq eps$$

Обозначим через v_i, $i = \overline{1, n}$ собственные значения матрицы $\tilde{A}^{-1}$. Далее, зная соотношения между матрицей и обратной к ней, получаем

$$\nu_i = \frac{1}{\lambda_i}, \quad \nu_{\max} = \nu_n = \frac{1}{\lambda_{\min}} = \frac{1}{\lambda_n}.$$

Таким образом минимальное собственное значение будет найдено по формуле: $\lambda_{min} = \frac{1}{v_{max}}$.

Собственный вектор x = y^k.

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)]
```

```python
    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
```

```python
    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:

A:

0.9546	-0.1256	0.0251	0.0502	0.1758
0.1306	1.4946	0.0000	-0.1005	0.1005
0.0754	0.0000	1.2007	-0.3517	0.2010
-0.1507	0.3165	0.0000	1.1806	-0.0502
0.6280	0.0000	0.2261	0.0251	1.4067

A.T @ A:

1.3511	0.0276	0.2565	-0.1539	1.0871
0.0276	2.3498	-0.0032	0.2171	0.1122
0.2565	-0.0032	1.4934	-0.4154	0.5638
-0.1539	0.2171	-0.4154	1.5308	-0.0959
1.0871	0.1122	0.5638	-0.0959	2.0627

Результаты:

1)Frobenius normal form:

-4.192	6.204	-4.144	1.269	-0.144
1.000	0.000	0.000	0.000	0.000
0.000	1.000	0.000	-0.000	0.000
-0.000	0.000	1.000	0.000	-0.000
-0.000	0.000	-0.000	1.000	-0.000

2)Characteristic polynomial:
p(x)=x^5 - 4.192x^4 + 6.204x^3 - 4.144x^2 + 1.269x - 0.144
3)r1 = p1 - SpA = -3.553e-14
4)r2 = p5 - detA = -7.966e-15
5)min eigenvalue: 0.5252455766731503
6)eigenvector:

-1.28167	-0.05444	-0.27667	-0.18735	1.00000

7)r of eigenvalue: -2.5274119394680383e-05
8)r of eigenvector:

-3.359e+00	-1.178e+00	4.723e-01	8.002e-02	-2.091e+00

9)count of iterations: 19

Danilevsky eigenvector:
5)min eigenvalue: 0.5252455766731503
6)eigenvector:

1.00000	0.52525	0.27588	0.14491	0.07611

7)r of eigenvalue: -2.5274119394680383e-05
8)r of eigenvector:

-5.845e-06	-1.592e-06	7.221e-07	2.022e-06	2.792e-06

9)count of iterations: 19

Для сравнения вычислим собственные значения при помощи стандартной библиотеки:

```
True eigenvalues: [0.52524796 1.05436889 1.64303265 2.41830293 3.14684458]
|lambda_1 / lambda_2| = 0.7684850235363426
```

Метод Данилевского является точным методом, это подтверждает маленькая погрешность. $|\lambda_1| = 3.14684458$, $|\lambda_2| = 2.41830293$, отсюда $\frac{|\lambda_2|}{|\lambda_1|} = 0.7684850235363426 < 1$ => 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 A. 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.