Чисельні методи в інформатиці
Лабораторна робота №3

Грищенко Юрій, ІПС-32, 2020

Нагадаємо постановку алгебричних задач на власні значення: для матриці A знайти такі λ й $\vec{x} \neq \vec{0}$, що $A\vec{x} - \lambda\vec{x} = 0$. Відшукання λ зводиться до розв'язання алгебричного рівняння $P_n(\lambda) = \det(A - \lambda E) = 0$. Воно має n коренів λ_i $i = \overline{1, n}$, яким відповідають власні вектори $\vec{x}_i : A\vec{x}_i = \lambda_i \vec{x}_i$. Задачу пошуку всіх власних значень і векторів називають *повною проблемою власних значень*. Якщо ж потрібно знайти тільки деякі з власних значень (наприклад, $\lambda_{\max}(A) = \max\limits_{i=\overline{1,n}} |\lambda_i|$, $\lambda_{\min}(A) = \min\limits_{i=\overline{1,n}} |\lambda_i|$ та інші), то її називають *частковою проблемою власних значень*.

$$A = \begin{pmatrix} 5.18+\alpha & 1.12 & 0.95 & 1.32 & 0.83 \\ 1.12 & 4.28-\alpha & 2.12 & 0.57 & 0.91 \\ 0.95 & 2.12 & 6.13+\alpha & 1.29 & 1.57 \\ 1.32 & 0.57 & 1.29 & 4.57-\alpha & 1.25 \\ 0.83 & 0.91 & 1.57 & 1.25 & 5.21+\alpha \end{pmatrix},$$

Для мого варіанту α=1,5.

Завдання 1.

Степеневим методом з точністю $\varepsilon=10^{-4}$ знайти максимальне та мінімальне за модулем власні значення та відповідні власні вектори заданої матриці.

Нехай власні значення впорядковано за зростанням їх модулів:

$$|\lambda_1| > |\lambda_2| \ge |\lambda_3| \ge \dots \ge |\lambda_n|.$$

мації (underflow). Тому потрібно нормувати вектори $\vec{x}^k$.

Алгоритм відшукання λ_1, $\vec{x}_1$ степеневим методом за формулою скалярних добутків (6.1) із нормуванням $\vec{x}^n$ має такий вигляд:

1) задати $\vec{x}^0$;

2) для $k = 0, 1, \dots$ обчислити $\vec{e}^k = \dfrac{\vec{x}^k}{\|\vec{x}^k\|}$, $\vec{x}^{k+1} = A\vec{e}^k$, $\mu_k = (\vec{x}^{k+1}, \vec{e}^k)$.

3) продовжити процес до виконання умови $|\mu_N - \mu_{N-1}| < \varepsilon$, де ε — задана точність.

Тоді $\lambda_1 \approx \mu_N$, $\vec{x}_1 \approx \vec{e}^N$.

Цей алгоритм реалізували на мові Python, використовуючи бібліотеку NumPy.

За 6 ітерацій знаходимо найбільше за модулем власне число та відповідний власний вектор:

10.880243244420626

```
[0.41548174 0.3021174  0.65540119 0.27759217 0.4790531 ]
```

Що приблизно співпадає з результатами алгоритму `np.linalg.eig` (наведені нижче)

Є декілька способів знаходження найменшого власного числа:

Якщо відоме $\lambda_{max}(A) = \lambda_1$, то утворимо матрицю $B = \lambda_1 E - A$, де E — одинична матриця. За допомогою степеневого метода знайде-

мо $\lambda_{max}(B)$. Тоді $\lambda_{max}(B) = \lambda_{max}(A) - \lambda_{min}(A)$ й $\lambda_{min}(A) = \lambda_{max}(A) - \lambda_{max}(B)$.

За 26 ітерацій знаходимо $\boldsymbol{\lambda_{min}}$ **= 1.8418565770988433**

Якщо $\lambda_{max}(A) = \lambda_1$ невідоме, то утворимо матрицю $B = \|A\|_\infty E - A$, де $\|A\|_\infty = \max_i \sum_j |a_{ij}|$. Оскільки $\lambda_1 \le \|A\|_\infty$, то $\lambda_{min}(A) = \|A\|_\infty - \lambda_{max}(B)$.

За 19 ітерацій знаходимо **λ_{min} = 1.8393337241812056**

Також найменше власне число та відповідний власний вектор можна знайти, використавши обернену матрицю:

спектрального кольца). Для этого достаточно степенной метод применить к обратной матрице A^{-1}, т. е. получить $|\lambda_{\max}(A^{-1})|$ и взять обратную величину:

$$\lambda_{\min}(A) = \frac{1}{\lambda_{\max}(A^{-1})}.$$

Соответствующий собственный вектор x^n будет

$$x^n \approx (A^{-1})^k y^{(0)} [\lambda_{\min}(A)]^k, \quad \left(y^{(0)}\right)^T = (11 \ \dots \ 1),$$

За 8 ітерацій знаходимо **λ_{min} = 1.838038567946642** та відповідний власний вектор:

```
[-0.1874288   0.89981521 -0.3218449   0.2101469  -0.08635491]
```

Завдання 2.

Методом обертання Якобі з точністю $\varepsilon=10^{-4}$ знайти всі власні значення та відповідні власні вектори заданої симетричної матриці.

Розв'язання повної проблеми власних значень. Для симетричної матриці $A = A^{\mathrm{T}}$ можна застосовувати ітераційний метод обертання (Якобі). Він полягає у виконанні ортогональних перетворень вихідної матриці A, які зводять її до діагонального вигляду $\Lambda = UAU^{\mathrm{T}}$, $\Lambda = \operatorname{diag}(\lambda_i)$, $U^{\mathrm{T}} = U^{-1}$, де λ_i — власні значення матриці A.

Побудуємо послідовність матриць $\{A_k\}$, що збігається до Λ, за правилом $A_{k+1} = U_k A_k U_k^{\mathrm{T}}$, $A_0 = A$, де

$$U_k = \begin{array}{c} \left(\begin{array}{ccccccccc} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \cos\varphi & \dots & \sin\varphi & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & -\sin\varphi & \dots & \cos\varphi & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{array} \right) \begin{array}{c} \\ \\ i \\ \\ j \\ \\ \\ \end{array} \\ \begin{array}{ccccccc} & & i & & j & & \end{array} \end{array}$$

елементарна матриця обертання. Її будують за таким алгоритмом. Виберемо в матриці A_k найбільший за модулем недіагональний елемент a_{ij}^k. Тоді

$$a_{ij}^{k+1} = a_{ij}^k \cos(2\varphi_k) + \frac{1}{2}\left(a_{jj}^k - a_{ii}^k\right)\sin(2\varphi_k).$$

Виберемо φ_k так, щоб $a_{ij}^{k+1} = 0$. Тоді $\operatorname{tg}(2\varphi_k) = \dfrac{2a_{ij}^k}{a_{ii}^k - a_{jj}^k} \equiv h_k$. Отже, $\varphi_k = \dfrac{1}{2}\operatorname{arctg} h_k$. Якщо $a_{ii}^k = a_{jj}^k$, то $\varphi_k = \dfrac{\pi}{4}$.

Нехай $t(A_k) = \sum\limits_{\substack{i,j=1 \\ i \neq j}}^{n} a_{ij}^2$. Виконується рівність $t(A_{k+1}) = t(A_k) - 2\left(a_{ij}^k\right)^2$.

Отже, ітераційний метод Якобі збігається зі швидкістю геометричної прогресії, знаменник якої q залежить від n. Ітераційний процес припиняється, якщо виконано умову $t(A_N) \leq \varepsilon$, де ε — точність обчислення власних значень. j-й стовпець номера матриці $\bar{U} = \prod\limits_{k=1}^{N} U_k$ дає наближення до власного вектора, що відповідає λ_j.

За 16 кроків знаходимо такі власні числа:

```
[ 6.1626021   1.83487542 10.88064944  2.38310941  5.60876362]
```

І відповідні власні вектори (по стовпчиках):

```
[[ 0.85940814 -0.18737756  0.4104302  -0.17382114 -0.16625452]
 [-0.03144835  0.90045447  0.30354031 -0.2775756  -0.13786882]
 [-0.48412205 -0.32278437  0.65815557 -0.03590845 -0.476422  ]
 [ 0.11191231  0.20603806  0.27796048  0.92956339  0.06061219]
 [-0.11631909 -0.08619722  0.47851808 -0.16541008  0.85011756]]
```

Результати, отримані в обох завданнях, приблизно співпадають з результатами вбудованого в NumPy алгоритму np.linalg.eig:

```
[10.88067536  1.83485572  2.38309792  6.16260149  5.60876952]
```

```
[[ 0.41116707  0.18677373  0.17366132 -0.85925229 -0.16608574]
 [ 0.30244399 -0.90087132  0.27770445  0.03156313 -0.137268  ]
 [ 0.65850416  0.32171141  0.03592527  0.48438034 -0.47640225]
 [ 0.27692596 -0.2066338  -0.92982757 -0.11181509  0.05943531]
 [ 0.47869986  0.08573614  0.16386593  0.11645752  0.85034191]]
```