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 153502
Богданов Александр Сергеевич

Проверил: Анисимов Владимир Яковлевич

Минск 2022

Содержание

Вариант 3

Цели выполнения задания:

1) Освоить методы вычисления собственных значений и векторов с помощью метода вращений.
2) Исследовать количество итераций для данного метода
3) Проверить правильность работы программы на тестовых примерах
4) Численно решить нелинейное уравнение заданного варианта
5) Напечать отчёт
6) Принести отчёт на сдачу

Краткие теоретические сведения

Метод Якоби — итерационный алгоритм для вычисления собственных значений и собственных векторов вещественной симметричной матрицы. Карл Густав Якоб Якоби, в честь которого назван этот метод, предложил его в 1846 году. Однако использоваться метод начал только в 1950-х годах с появлением компьютеров.

Метод Якоби (вращений) использует итерационный процесс, который приводит исходную симметрическую матрицу A к диагональному виду с помощью последовательности *элементарных ортогональных преобразований* (в дальнейшем называемых *вращениями Якоби* или *плоскими вращениями).* Процедура построена таким образом, что на *(k+1)*-ом шаге осуществляется преобразование вида

$$A^{(k)} \rightarrow A^{(k+1)} = V^{(k)*} A^{(k)} V^{(k)} = V^{(k)*} \dots V^{(0)*} A^{(0)} V^{(0)} \dots V^{(k)}, \ k=0,1,2\dots, \qquad (5.1)$$

где $A^{(0)} = A$, $V^{(k)} = V^{(k)}_{ij}(\varphi)$ — ортогональная матрица, отличающаяся от единичной матрицы только элементами

$$v_{ii} = v_{jj} = \cos\varphi, \quad v_{ij} = -v_{ji} = -\sin\varphi\ , \qquad (5.2)$$

значение φ выбирается при этом таким образом, чтобы обратить в 0 наибольший по модулю недиагональный элемент матрицы $A^{(k)}$. Итерационный процесс постепенно приводит к матрице со значениями недиагональных элементов, которыми можно пренебречь, т.е. матрица $A^{(k)}$ все более похожа на диагональную, а диагональная матрица Λ является пределом последовательности $A^{(k)}$ при $k \rightarrow \infty$.

Алгоритм метода вращений.

1) В матрице $A^{(k)}$ $(k=0,1,2,\dots)$ среди всех недиагональных элементов выбираем максимальный по абсолютной величине элемент, стоящий выше главной диагонали; определяем его номера i и j строки и столбца, в которых он стоит (если максимальных элементов несколько, можно взять любой из них);

2) По формулам

$$\cos\varphi_k = \cos\left(\frac{1}{2}\cdot \operatorname{arctg} P_k\right),$$
$$\sin\varphi_k = \sin\left(\frac{1}{2}\cdot \operatorname{arctg} P_k\right),$$
$$\text{где } P_k = \begin{cases} \dfrac{\pi}{4}, & \text{если } a_{ii}^{(k)} = a_{jj}^{(k)} \\ \dfrac{2\,a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}}, & \text{иначе} \end{cases}$$

вычисляем $\cos\varphi_k$ и $\sin\varphi_k$, получаем матрицу $V^{(k)} = V^{(k)}_{ij}(\varphi_k)$

$$V^{(k)} = \begin{array}{cc} & \begin{array}{ccccccccc} & & i & & & & j & & \end{array} \\ \begin{array}{c} \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \end{array} & \begin{bmatrix} 1 & \dots & \dots & \dots & 0 & \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \cos\varphi_k & \dots & \dots & \dots & -\sin\varphi_k & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & 1 & \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \sin\varphi_k & \dots & \dots & \dots & \cos\varphi_k & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & 0 & \dots & \dots & \dots & 1 \end{bmatrix} \end{array}$$

3) По формуле

$$A^{(k+1)} = V^{(k)\mathrm{T}} \cdot A^{(k)} \cdot V^{(k)}$$

находим матрицу $A^{(k+1)}$.

4) Итерационный процесс останавливаем, когда в пределах принятой точности суммой квадратов всех недиагональных элементов матрицы $A^{(k+1)}$ можно пренебречь.

5) В качестве собственных значений матрицы A берем диагональные элементы матрицы $A^{(k+1)}$, в качестве собственных векторов – соответствующие столбцы матрицы

$$V = V^{(0)} V^{(1)} \dots V^{(k)}.$$

Основное достоинство метода Якоби заключается в том, что при выполнении каждого плоского вращения уменьшается сумма квадратов недиагональных элементов; сходимость этой суммы к нулю по мере увеличения числа шагов гарантирует сходимость процесса диагонализации.

Наиболее простой метод поиска собсвенных значений:

Пусть число λ и вектор $x \in L, x \neq 0$ таковы, что

$$Ax = \lambda x. \qquad (1)$$

Тогда число λ называется собственным числом линейного оператора A, а вектор x собственным вектором этого оператора, соответствующим собственному числу λ.

В конечномерном пространстве L_n векторное равенство (1) эквивалентно матричному равенству

$$(A - \lambda E)X = 0, \quad X \neq 0. \qquad (2)$$

Отсюда следует, что число λ есть собственное число оператора A в том и только том случае, когда детерминант $det(A - \lambda E) = 0$, т. е. λ есть корень многочлена $p(\lambda) = det(A - \lambda E)$, называемого характеристическим многочленом оператора A. Столбец координат X любого собственного вектора соответствующего собственному числу λ есть нетривиальное решение однородной системы (2).

Программная реализация.

Вспомогательные функции

```
def print_array(matrix, msg="", sep='\n'):
    n = matrix.shape[0]
    m = None

    if msg != "":
        print(msg)
    if len(matrix.shape) == 2:
        m = matrix.shape[1]

    if len(matrix.shape) == 2:
        for i in range(n):
            for j in range(m):
                print('{:>8.5f}'.format(matrix[i, j]), end=' ')
            print(sep, end='')

    elif len(matrix.shape) == 1:
        for i in range(n):
            print('{:>8.5f}'.format(matrix[i]), end=' ')
        print(sep, end='')
```

```
def symmetric_check(matrix, eps):
    for i in range(1, len(matrix)):
        for j in range(i):
            if (abs(matrix[i][j] - matrix[j][i]) >= eps):
                raise ValueError("Матрица должна быть симметричной!")
```

Вспомогательные функции для метода вращений

```
def check_equal_dim(matrix):
     if matrix.shape[0] != matrix.shape[1]:
        raise ValueError("Матрица размера не NxN")
```

```
def calc_non_diag(matrix):
    check_equal_dim(matrix)

    n = matrix.shape[0]
    sum_n_diag = 0
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            sum_n_diag += abs(matrix[i, j])
    return sum_n_diag
```

```
def max_no_diag(matrix):
    check_equal_dim(matrix)

    n = matrix.shape[0]
    max_val = matrix[0, 1]
    val_row = 0
    val_col = 1
    for i in range(n):
        for j in range(i + 1, n):
            if abs(matrix[i, j]) > abs(max_val):
                max_val = matrix[i, j]
                val_row = i
                val_col = j

    return max_val, val_row, val_col
```

Метод вращений

```
def find_eigen(matrix, err):
    check_equal_dim(matrix)
    symmetric_check(matrix, err)
    A = matrix.copy()
    n = A.shape[0]
    rotate_matrix = np.eye(n)  # diagonal matrix
    eig_vec = np.zeros(shape=A.shape)
    iteration = 0
    while calc_non_diag(A) > err:

        max_el, p, q = max_no_diag(A)

        if A[p, p] == A[q, q]:
            if max_el > 0:
                teta = np.pi / 4
            else:
                teta = -1 * np.pi / 4
        else:
            teta = np.arctan((2 * max_el) / (A[p, p] - A[q, q])) / 2

        # fill rotate matrix
        rotate_matrix = np.eye(n)
        rotate_matrix[p, p] = np.cos(teta)
        rotate_matrix[q, q] = np.cos(teta)
        rotate_matrix[p, q] = np.sin(teta) * -1
        rotate_matrix[q, p] = np.sin(teta)

        A = rotate_matrix.T @ A @ rotate_matrix

        if iteration != 0:
            eig_vec = eig_vec @ rotate_matrix
        else:
            eig_vec = rotate_matrix.copy()

        iteration += 1

    return A, eig_vec, iteration
```

Полученные результаты

Проверять примеры будем при помощи пакета numpy.

Тестовый пример 1.

С точностью 0.0001 вычислить собственные значения и собственные вектора матрицы A.

$$A = \begin{bmatrix} 5 & 1 & 2 \\ 1 & 4 & 1 \\ 2 & 1 & 3 \end{bmatrix}$$

```
ВЫПОЛНЯЮТСЯ ВЫЧИСЛЕНИЯ МЕТОДОМ ВРАЩЕНИЙ...

Матрица собственных значений:
 6.89511  0.00000  0.00000
 0.00000  3.39730  0.00000
-0.00000  0.00000  1.70760
Собственные значения:
 6.89511  3.39730  1.70760
Собственные вектора:
 0.75258 -0.45794 -0.47320
 0.43170  0.88574 -0.17060
 0.49725 -0.07589  0.86428
Количество итераций: 7

Проверка с помощью numpy:

Собственные значения (numpy):
 6.89511  3.39730  1.70760
Собственные вектора (numpy):
-0.75258 -0.45794 -0.47320
-0.43170  0.88574 -0.17060
-0.49725 -0.07589  0.86428
```

Тестовый пример 2.

С точностью 0.0001 вычислить собственные значения и собственные вектора матрицы А.

$$A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

```
Матрица согласно условию:
 1.00000 -3.00000  2.00000
-3.00000  0.00000  1.00000
 2.00000  1.00000  1.00000

ВЫПОЛНЯЮТСЯ ВЫЧИСЛЕНИЯ МЕТОДОМ ВРАЩЕНИЙ...

Матрица собственных значений:
 3.88824  0.00000  0.00000
-0.00000 -3.50331  0.00000
 0.00000  0.00000  1.61507
Собственные значения:
 3.88824 -3.50331  1.61507
Собственные вектора:
 0.78029  0.62496 -0.02384
-0.50834  0.65598  0.55792
 0.36432 -0.42322  0.82955
Количество итераций: 7

Проверка с помощью numpy:

Собственные значения (numpy):
-3.50331  3.88824  1.61507
Собственные вектора (numpy):
-0.62496 -0.78029 -0.02384
-0.65598  0.50834  0.55792
 0.42322 -0.36432  0.82955
```

Тестовый пример 3.

С точностью 0.0001 вычислить собственные значения и собственные вектора матрицы А.

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

```
ВЫПОЛНЯЮТСЯ ВЫЧИСЛЕНИЯ МЕТОДОМ ВРАЩЕНИЙ...

Матрица собственных значений:
 3.00000  0.00000  0.00000  0.00000
 0.00000  1.00000  0.00000  0.00000
 0.00000  0.00000  1.00000  0.00000
 0.00000  0.00000  0.00000 -1.00000
Собственные значения:
 3.00000  1.00000  1.00000 -1.00000
Собственные вектора:
 0.50000 -0.50000 -0.50000  0.50000
 0.50000  0.50000 -0.50000 -0.50000
 0.50000 -0.50000  0.50000 -0.50000
 0.50000  0.50000  0.50000  0.50000
Количество итераций: 4

Проверка с помощью numpy:

Собственные значения (numpy):
-1.00000  1.00000  3.00000  1.00000
Собственные вектора (numpy):
 0.50000 -0.70711 -0.50000 -0.01305
-0.50000 -0.00000 -0.50000  0.70699
-0.50000  0.00000 -0.50000 -0.70699
 0.50000  0.70711 -0.50000  0.01305
```

Тестовый пример 4:

Попробуем наш алгоритм для несимметричной матрицы:

```
A = np.array([
    [11, 11, 1, 0],
    [1, 1, 0, 1],
    [1, 0, 1, 1],
    [0, 1, 1, 1]
])

solve(A, E)
```

```
Матрица должна быть симметричной!
Проводится решение другим алгоритмом...
Собственные значения:
12.09902 -1.00000  1.90098  1.00000
Собственные вектора:
-0.99158 -0.50000 -0.44798 -0.50000
-0.09081  0.50000  0.33968  0.50000
-0.09081  0.50000  0.33968 -0.50000
-0.01636 -0.50000  0.75402  0.50000
```

Решение задания:

Вариант 3.

С точностью 0.0001 вычислить собственные значения и собственные вектора матрицы A.

$$C=\begin{bmatrix}0,2 & 0 & 0,2 & 0 & 0\\ 0 & 0,2 & 0 & 0,2 & 0\\ 0,2 & 0 & 0,2 & 0 & 0,2\\ 0 & 0,2 & 0 & 0,2 & 0\\ 0 & 0 & 0,2 & 0 & 0,2\end{bmatrix},\quad D=\begin{bmatrix}2,33 & 0,81 & 0,67 & 0,92 & -0,53\\ 0,81 & 2,33 & 0,81 & 0,67 & 0,92\\ 0,67 & 0,81 & 2,33 & 0,81 & 0,92\\ 0,92 & 0,67 & 0,81 & 2,33 & -0,53\\ -0,53 & 0,92 & 0,92 & -0,53 & 2,33\end{bmatrix}.$$

где A = kC + D, A – исходная матрица для расчёта, k – номер варианта (0-15), матрицы C, D заданы ниже:

```
Матрица согласно условию:
 2.93000  0.81000  1.27000  0.92000 -0.53000
 0.81000  2.93000  0.81000  1.27000  0.92000
 1.27000  0.81000  2.93000  0.81000  1.52000
 0.92000  1.27000  0.81000  2.93000 -0.53000
-0.53000  0.92000  1.52000 -0.53000  2.93000

ВЫПОЛНЯЮТСЯ ВЫЧИСЛЕНИЯ МЕТОДОМ ВРАЩЕНИЙ...

Матрица собственных значений:
 2.50483 -0.00000 -0.00000  0.00000  0.00000
-0.00000  6.06293  0.00000  0.00000 -0.00000
-0.00000  0.00000  4.08408  0.00000  0.00000
 0.00000  0.00000  0.00000  1.60465  0.00000
 0.00000 -0.00000  0.00000 -0.00000  0.39352
Собственные значения:
 2.50483  6.06293  4.08408  1.60465  0.39352
Собственные вектора:
 0.56083  0.43337 -0.39300 -0.44298  0.38339
-0.58447  0.50629  0.02368 -0.54518 -0.32295
 0.43481  0.54770  0.26768  0.37704 -0.54512
-0.38669  0.42867 -0.44712  0.60357  0.32017
-0.07269  0.26857  0.75725  0.01025  0.59081
Количество итераций: 28

Проверка с помощью numpy:

Собственные значения (numpy):
 6.06293  4.08408  0.39352  2.50483  1.60465
Собственные вектора (numpy):
 0.43337  0.39300 -0.38339 -0.56083 -0.44298
 0.50629 -0.02368  0.32295  0.58447 -0.54518
 0.54770 -0.26768  0.54512 -0.43481  0.37704
 0.42867  0.44711 -0.32017  0.38669  0.60357
 0.26857 -0.75725 -0.59081  0.07269  0.01025
```

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.

Из примеров можно заметить, что данный алгоритм имеет малое количество итераций, что позволяет решать поставленные примеры достаточно быстро. Разобраны 4 тестовых примера, которые говорят о том, что алгоритм корректно отрабатывает для любой симметричной матрицы, а также проведена сверка с numpy методом для определения собственных векторов и значений. Была проведена проверка на симметричность матрицы. Проверка на сходимость не требуется, так как данный метод сходится всегда. В программе был предусмотрен другой метод решения при несимметричности исходной матрицы.