

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»
Кафедра информатики

Отчет по лабораторной работе №5
Нахождение собственных значений и собственных векторов

Выполнил:
студент группы 953501
Кременевский В.С

Руководитель:
доцент
Анисимов В.Я.

Минск 2021

Содержание

- 1. Цель работы**
- 2. Теоретические сведения**
- 3. Тестовые примеры**
- 4. Решение задания**
- 5. Выводы**
- 6. Программная реализация**

1.

Цель работы

- 1) Рассмотреть методы вычисления собственных значений и векторов
- 2) Реализовать один из методов программно
- 3) Рассмотреть и разобраться с матрицами поворота и как они действуют на вектор
- 4) Освоить метод вращений Якоби для вычисления собственных значений и векторов симметричной матрицы
- 5) Научиться определить подобие матриц в результате преобразования

2.

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.

Матрица поворота в двумерном пространстве

В двумерном пространстве поворот можно описать одним углом θ со следующей матрицей линейного преобразования в декартовой системе координат:

$$M(\theta) = \begin{pmatrix} \cos \theta & \mp \sin \theta \\ \pm \sin \theta & \cos \theta \end{pmatrix}$$

Поворот выполняется путём умножения матрицы поворота на вектор-столбец, описывающий вращаемую точку:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \mp \sin \theta \\ \pm \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Координаты (x', y') в результате поворота точки (x, y) имеют вид:

$$\begin{aligned} x' &= x \cos \theta \mp y \sin \theta, \\ y' &= \pm x \sin \theta + y \cos \theta. \end{aligned}$$

Конкретные знаки в формулах зависят от того, является ли система координат правосторонней или левосторонней, и выполняется ли вращение по или против часовой стрелки. Верхний знак указан для обычного соглашения: правосторонняя система координат и положительное направление вращения против часовой стрелки (тот же знак верен для левосторонней координатной системы при выборе положительного направления вращения по часовой стрелке; в оставшихся двух комбинациях — нижний знак).

Матрица поворота в 3-мерном пространстве

Любое вращение в трёхмерном пространстве может быть представлено как композиция поворотов вокруг трёх ортогональных осей (например, вокруг осей декартовых координат). Этой композиции соответствует матрица, равная произведению соответствующих трёх матриц поворота.

Матрицами вращения вокруг оси [декартовой системы координат](#) на угол α в трёхмерном пространстве с неподвижной системой координат являются:

- Вращение вокруг оси x :

$$M_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{pmatrix}.$$

- Вращение вокруг оси y :

$$M_y(\alpha) = \begin{pmatrix} \cos \alpha & 0 & \sin \alpha \\ 0 & 1 & 0 \\ -\sin \alpha & 0 & \cos \alpha \end{pmatrix}.$$

- Вращение вокруг оси z :

$$M_z(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Где M_x - поворот осей YZ , X - неподвижна, M_y - поворот осей XZ , Y - неподвижна, M_z - поворот осей XY , Z - неподвижна

Матрица поворота в n -мерном пространстве

$$M_{1,2}(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

— матрица поворота в 5-мерном пространстве в плоскости x_1x_2 ,

$$M_{2,4}(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \cos \alpha & 0 & -\sin \alpha & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sin \alpha & 0 & \cos \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

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

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

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

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

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 A 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

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

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

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

3) По формуле

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

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.

$$J_i = \begin{bmatrix} & & j & & k \\ & 1 & & & \\ & & \ddots & & \\ & & & \cos(\theta) & -\sin(\theta) \\ j & & & & \ddots \\ & & & \sin(\theta) & \cos(\theta) \\ & & & & \ddots \\ k & & & & 1 \\ & & & & 1 \end{bmatrix}$$

Суть алгоритма заключается в том, чтобы для заданной симметрической матрице $A = A^{(0)}$ построить последовательность ортогонально подобных матриц $A^{(1)}, A^{(2)}, \dots, A^{(m)}$, сходящуюся к диагональной матрице, на диагонали которой стоят собственные значения A . Для построения этой последовательности применяется специально подобранная матрица вращения J_i , такая что норма наддиагональной части $\|A^{(i)}\|_{off} = \sqrt{\sum_{1 \leq j < k \leq n} (a_{jk}^{(i)})^2}$ уменьшается при каждом двустороннем вращении матрицы $A^{(i+1)} = J_i^T A^{(i)} J_i$.

1.2 Математическое описание

Исходные данные: симметрическая матрица A (элементы a_{ij}).

Вычисляемые данные: диагональная матрица Λ (элементы λ_{ij}).

Норма наддиагональной части $\|A^{(i)}\|_{off} = \sqrt{\sum_{1 \leq j < k \leq n} (a_{jk}^{(i)})^2}$

уменьшается при каждом двустороннем вращении матрицы $A^{(i+1)} = J_i^T A^{(i)} J_i$.

Это достигается обнулением^[1] элемента матрицы $A^{(i)}$ в матрице $A^{(i+1)}$. Если он расположен в j -й строке и k -м столбце, то

Если обозначить $s = \sin \theta$ и $c = \cos \theta$, то матрица $A^{(i+1)}$ состоит из следующих элементов, отличающихся от элементов $A^{(i)}$:

$$\begin{aligned} a_{jj}^{(i+1)} &= c^2 a_{jj}^{(i)} - 2sc a_{jk}^{(i)} + s^2 a_{kk}^{(i)} \\ a_{kk}^{(i+1)} &= s^2 a_{jj}^{(i)} + 2sc a_{jk}^{(i)} + c^2 a_{kk}^{(i)} \\ a_{jk}^{(i+1)} &= a_{kj}^{(i+1)} = (c^2 - s^2) a_{jk}^{(i)} + sc (a_{kk}^{(i)} - a_{jj}^{(i)}) \\ a_{jm}^{(i+1)} &= a_{mj}^{(i+1)} = c a_{jm}^{(i)} - s a_{km}^{(i)} & m \neq j, k \\ a_{km}^{(i+1)} &= a_{mk}^{(i+1)} = s a_{jm}^{(i)} + c a_{km}^{(i)} & m \neq j, k \\ a_{ml}^{(i+1)} &= a_{ml}^{(i)} & m, l \neq j, k \end{aligned}$$

Можно выбрать θ так, чтобы $a_{jk}^{(i+1)} = 0$ и $a_{kj}^{(i+1)} = 0$. Отсюда следует равенство

$$\frac{a_{jj}^{(i)} - a_{kk}^{(i)}}{2a_{jk}^{(i)}} = \frac{c^2 - s^2}{2sc} = \frac{\cos(2\theta)}{\sin(2\theta)} = \operatorname{ctg}(2\theta) \equiv \tau.$$

Если $a_{jj}^{(i)} = a_{kk}^{(i)}$, то выбирается $\theta = \frac{\pi}{4}$, в противном случае вводится $t = \frac{s}{c} = \operatorname{tg}(\theta)$ и тогда $t^2 - 2t\tau + 1 = 0$. Решение квадратного уравнения даёт $t = \frac{\operatorname{sign}(\tau)}{|\tau| + \sqrt{1 + \tau^2}}$, $c = \frac{1}{\sqrt{1 + t^2}}$, $s = tc$.

Вычисление останавливается, когда выполняются критерии близости к диагональной матрице. Это малость максимального по абсолютной величине внедиагонального элемента матрицы.

Сходимость данного метода можно установить путем вычисления нормы Фробениуса не диагональных элементов, которая уменьшается для матрицы с каждым шагом.

$$\begin{aligned} off(A')^2 &= ||A'||_F^2 - \sum_{i=1}^n a'_{ii}^2 \\ &= ||A||_F^2 - \sum_{i \neq p,q}^n a'_{ii}^2 - (a'_{pp}^2 + a'_{qq}^2) \\ &= ||A||_F^2 - \sum_{i \neq p,q}^n a_{ii}^2 - (a_{pp}^2 + 2a_{pq}^2 + a_{qq}^2) \\ &= ||A||_F^2 - \sum_{i=1}^n a_{ii}^2 - 2a_{pq}^2 \\ off(A')^2 &= off(A)^2 - 2a_{pq}^2 \\ off(A')^2 &< off(A)^2 \end{aligned}$$

Причем так как мы постоянно получаем подобную матрицу, путем умножения слева и справа на матрицу поворота, норма фробениуса, должна оставаться постоянной, при том что все кроме диагональных элементов уменьшаются, это значит, что “центр тяжести матрицы” просто якобы смещается на диагональ, но не изменяется.

Это можно проверить:

```
Computing eigenvalues...
Frabenous norm: 12.155603646055592
1 step:
    off(A) = 54.0142
    off(A') = 36.9614
Frabenous norm: 12.15560364605559
2 step:
    off(A) = 36.9614
    off(A') = 22.7036
Frabenous norm: 12.155603646055587
3 step:
    off(A) = 22.7036
    off(A') = 12.4636
Frabenous norm: 12.15560364605559
4 step:
    off(A) = 12.4636
    off(A') = 8.7183
Frabenous norm: 12.15560364605559
5 step:
    off(A) = 8.7183
    off(A') = 3.9293
Frabenous norm: 12.15560364605559
6 step:
    off(A) = 3.9293
    off(A') = 2.9636
Frabenous norm: 12.15560364605559
```

Как видно норма фробениуса остается почти незменной на каждой итерации.

Наиболее простой метод поиска собственных значений:

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

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

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

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

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).

Метод Данилевского

Метод Данилевского относится к прямым методам и является достаточно простым и экономичным. Известно, что матрицы $S^{-1}AS$, полученные преобразованием подобия из с A , имеют тот же характеристический многочлен, что и A . Известно также, что любая матрица приводима преобразованием подобия к так называемой канонической форме Фробениуса

$$F = \begin{pmatrix} -p_1 & -p_2 & \dots & -p_n \\ 1 & 0 & & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & & & & \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix},$$

в первой строке которой стоят коэффициенты характеристического многочлена, взятые с обратным знаком. Таким образом, основная задача сводится к нахождению матрицы S такой, что $F = S^{-1}AS$.

Предположим, что элемент a_{nn-1} матрицы A отличен от нуля. Разделим $(n-1)$ -й столбец этой матрицы на a_{nn-1} и вычтем его из i -го столбца, умноженного на a_{ni} (для всех $i=1, 2, \dots, n$). Тогда последняя строка примет такой же вид как в матрице F . Непосредственно

проверяется, что проделанная операция равносильна умножению A справа на матрицу

$$M_{n-1} = \begin{pmatrix} 1 & 0 & & & \dots & 0 \\ 0 & 1 & 0 & & & \dots & 0 \\ & & \dots & & & & \\ -\frac{a_{n1}}{a_{nn-1}} & & -\frac{a_{nn-2}}{a_{nn-1}} & \frac{1}{a_{nn-1}} & & -\frac{a_{nn}}{a_{nn-1}} & \\ 0 & & 0 & 0 & & & 1 \end{pmatrix}.$$

Непосредственно проверяется также, что M_{n-1} не вырождена и, следовательно, существует

$$M^{-1}_{n-1} = \begin{pmatrix} 1 & 0 & & & \dots & 0 \\ 0 & 1 & 0 & & & \dots & 0 \\ & & \dots & & & & \\ a_{n1} & & & \dots & & a_{nn} \\ 0 & & & 0 & 1 & 0 \end{pmatrix}.$$

Очевидно, что умножение AM_{n-1} слева на матрицу M^{-1}_{n-1} не меняет последней строки матрицы AM_{n-1} . Таким образом,

$$M^{-1}_{n-1} AM_{n-1} = A^{(1)} = \begin{pmatrix} a_{11}^{(1)} & & & & \dots & a_{1n}^{(1)} \\ a_{21}^{(1)} & & & & & a_{2n}^{(1)} \\ & \dots & & & & \\ a_{n-1,1}^{(1)} & & & & a_{n-1,n-1}^{(1)} & a_{n-1,n}^{(1)} \\ 0 & & & 0 & 1 & 0 \end{pmatrix}.$$

Заметим, что матрицы M_{n-1} и M^{-1}_{n-1} , умножением на которые мы переходим от матрицы A к матрице $A^{(1)}$, записываются непосредственно по виду матрицы A . Предположим далее, что элемент $a_{n-1,n-2}^{(1)}$ тоже отличен от нуля. Делаем второй шаг, полностью аналогичный предыдущему, и приводим вторую строку матрицы к виду необходимому для формы Фробениуса (сохраняя последнюю строку без изменений). Получаем

$$M^{-1}_{n-2} M^{-1}_{n-1} AM_{n-1} M_{n-2} = M^{-1}_{n-2} A^{(1)} M_{n-2} =$$

$$= A^{(2)} = \begin{pmatrix} a_{11}^{(2)} & & & & \dots & a_{1n}^{(2)} \\ a_{21}^{(2)} & & & & & a_{2n}^{(2)} \\ & \dots & & & & \\ a_{n-2,1}^{(2)} & & & & a_{n-2,n-1}^{(2)} & a_{n-2,n}^{(2)} \\ 0 & & & 1 & 0 & 0 \\ 0 & & & 0 & 1 & 0 \end{pmatrix},$$

где

$$M_{n-2} = \begin{pmatrix} 1 & 0 & & & & 0 \\ 0 & 1 & 0 & & & 0 \\ & & & \ddots & & \\ -\frac{a_{n-11}^{(1)}}{a_{n-1n-2}^{(1)}} & -\frac{a_{n-1n-3}^{(1)}}{a_{n-1n-2}^{(1)}} & \frac{1}{a_{n-1n-2}^{(1)}} & -\frac{a_{n-1n-3}^{(1)}}{a_{n-1n-2}^{(1)}} & -\frac{a_{n-1n}^{(1)}}{a_{n-1n-2}^{(1)}} & \\ 0 & 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 0 & 1 & \end{pmatrix},$$

$$M^I_{n-2} = \begin{pmatrix} 1 & 0 & & & & 0 \\ 0 & 1 & 0 & & & 0 \\ & & & \ddots & & \\ a_{n-11}^{(1)} & a_{n-1n-1}^{(1)} & a_{n-1n}^{(1)} & & & \\ 0 & 0 & 1 & 0 & & \\ 0 & 0 & 0 & 0 & 1 & \end{pmatrix}.$$

Правило построения матриц M_{n-2} и M^I_{n-2} по виду матрицы $A^{(1)}$, как видим, полностью сохраняется. Оно сохраняется и на следующих шагах метода. Таким образом, если имеет место так называемый регулярный случай, когда

$$a_{nn-1} \neq 0, a_{n-1n-2}^{(1)} \neq 0, a_{n-2n-3}^{(2)} \neq 0, \dots, a_{21}^{(n-2)} \neq 0,$$

то после $(n-1)$ шагов метода Данилевского получим следующий результат

$$A^{(n-1)} = M^I_1 \dots M^I_{n-1} A M_{n-1} \dots M_1 = \\ = \begin{pmatrix} a_{11}^{(n-1)} & a_{12}^{(n-1)} & & & a_{1n}^{(n-1)} \\ 1 & 0 & & & 0 \\ 0 & 1 & 0 & & 0 \\ 0 & 0 & 1 & & 0 \\ & & & \ddots & \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} -p_1 & -p_2 & \dots & -p_n \\ 1 & 0 & & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ & & & \ddots \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = F = S^{-1} AS.$$

В таком случае мы можем непосредственно выписать характеристическое уравнение

$$\lambda^n + p_1 \lambda^{n-1} + \dots + p_n = 0$$

и, решая его, найти собственные значения $\lambda_1, \lambda_2, \dots, \lambda_n$.

Для нахождения собственных векторов в регулярном случае нет необходимости решать систему (4.2). Как уже говорилось выше, матрицы F и A имеют одни и те же собственные значения. Собственные векторы, отвечающие одному и тому же собственному значению, вообще говоря, будут разными. Однако они связаны между собой преобразованием подобия. Так, если \bar{y} собственный вектор матрицы F , отвечающий собственному значению λ , то вектор $S\bar{y}$ будет

собственным вектором матрицы A , отвечающим тому же собственному значению.

Действительно, поскольку $F\bar{y} = \lambda \bar{y}$ и $F = S^{-1}AS$, то $S^{-1}AS\bar{y} = \lambda \bar{y}$.

Умножая это равенство слева на матрицу S , получим $AS\bar{y} = \lambda S\bar{y}$.

Последнее означает, что $S\bar{y}$ будет собственным вектором матрицы A .

Таким образом, собственные векторы матрицы A находятся пересчетом собственных векторов матрицы Фробениуса. Собственные же векторы \bar{y} матрицы Фробениуса определяются из системы

$$\begin{pmatrix} -p_1 & -p_2 & \dots & -p_n \\ 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & & & & \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \lambda \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

По координатная запись этой системы имеет вид

$$-p_1 y_1 - p_2 y_2 - \dots - p_n y_n = \lambda y_1,$$

$$y_1 = \lambda y_2,$$

$$y_2 = \lambda y_3,$$

.....

$$y_{n-1} = \lambda y_n.$$

Поскольку собственный вектор определяется с точностью до постоянного множителя, можно принять $y_n = 1$ и вычислить остальные координаты собственного вектора:

$$y_n = 1, \quad y_{n-1} = \lambda, \dots, y_1 = \lambda^{n-1}.$$

Равенство же

$$-p_1 y_1 - p_2 y_2 - \dots - p_n y_n = \lambda y_1$$

принимает при этом тривиальный вид

$$\lambda^n + p_1 \lambda^{n-1} + \dots + p_n = 0$$

и используется для контроля вычислений.

Зная матрицу S , не трудно теперь найти собственные векторы матрицы A .

Отдельно рассмотрим нерегулярный случай метода Данилевского. Пусть выполнено $(n-k)$ шагов метода и оказалось, что в матрице $A^{(n-k)}$ элемент $a_{kk-1}^{(n-k)} = 0$. Тогда, если левее этого элемента в строке есть отличные от нуля элементы (например в столбце с номером j), то поменяем местами j -й и $(k-1)$ -й столбцы и продолжим процесс. Заметим, что операция замены столбцов местами равносильна умножению матрицы $A^{(n-k)}$ слева и справа на матрицу T , которая

строиться из единичной матрицы E заменой четырех ее элементов. Именно:

$$t_{jj} = t_{k-1k-1} = 0, \quad t_{jk-1} = t_{k-1j} = 1,$$

остальные элементы матрицы T совпадают с соответствующими элементами матрицы E . Таким образом, в цепочке преобразований матрицы на данном шаге добавится дополнительная операция

$$TA^{(n-k)}T,$$

после которой процесс пойдет, как и раньше. При этом важно, что дополнительное преобразование $TA^{(n-k)}T$ является преобразованием подобия. Действительно, поскольку двойная перестановка столбцов дает исходную матрицу, то $TT = T^2 = E$, т.е. $T^{-1} = T$.

Если левее элемента $a_{kk-1}^{(n-k)} = 0$ в строке матрицы $A^{(n-k)}$ не оказалось ненулевых элементов, то матрица $A^{(n-k)}$ очевидно имеет вид

$$A^{(n-k)} = \begin{pmatrix} B^{(n-k)} & C^{(n-k)} \\ 0 & F^{(n-k)} \end{pmatrix},$$

$$\text{где } B^{(n-k)} = \begin{pmatrix} a_{11}^{(n-k)} & \dots & a_{1k-1}^{(n-k)} \\ a_{21}^{(n-k)} & \dots & a_{2k-1}^{(n-k)} \\ \dots & & \dots \\ a_{k-11}^{(n-k)} & \dots & a_{k-1k-1}^{(n-k)} \end{pmatrix},$$

$$F^{(n-k)} = \begin{pmatrix} a_{kk}^{(n-k)} & \dots & a_{kn}^{(n-k)} \\ 1 & 0 & \dots & 0 & 0 \\ \dots & & & & \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}.$$

Тогда

$$|A^{(n-k)} - \lambda E| = |B^{(n-k)} - \lambda E_{k-1}| \cdot |F^{(n-k)} - \lambda E_{n-k+1}|$$

и, следовательно, поскольку $F^{(n-k)}$ является матрицей Фробениуса, ее характеристический многочлен можно выписать непосредственно, а к матрице $B^{(n-k)}$ применить снова метод Данилевского. Таким образом, вычислительный процесс даже упрощается.

Подсчетом необходимых арифметических операций можно убедиться, что метод Данилевского является одним из самых экономичных методов решения полной проблемы собственных значений. Однако этот метод очень чувствителен к ошибкам в результатах промежуточных вычислений.

Тестовый пример 1

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

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

```
Computing eigenvalues...
1 step:
    off(A) = 12.0000
    off(A') = 4.0000
2 step:
    off(A) = 4.0000
    off(A') = 0.2111
3 step:
    off(A) = 0.2111
    off(A') = 0.0390
4 step:
    off(A) = 0.0390
    off(A') = 0.0012
5 step:
    off(A) = 0.0012
    off(A') = 0.0000
6 step:
    off(A) = 0.0000
    off(A') = 0.0000
Stopping computing...

Default Matrix:
  5.0000  1.0000  2.0000
  1.0000  4.0000  1.0000
  2.0000  1.0000  3.0000
```

```
Matrix of Eigenvalues:
  6.8951  0.0000 -0.0000
  0.0000  3.3973  0.0000
 -0.0000 -0.0000  1.7076
Eigenvalues:
  6.8951  3.3973  1.7076
Eigenvectors:
  0.7526 -0.4579 -0.4732
  0.4317  0.8857 -0.1706
  0.4973 -0.0759  0.8643

Verification:
Eigenvalues (numpy):
  6.8951  3.3973  1.7076
Eigenvectors (numpy):
 -0.7526 -0.4579 -0.4732
 -0.4317  0.8857 -0.1706
 -0.4973 -0.0759  0.8643
```

Тестовый пример 2

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

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

```
Computing eigenvalues...
1 step:
  off(A) = 4.0000
  off(A') = 2.0000
2 step:
  off(A) = 2.0000
  off(A') = 0.5528
3 step:
  off(A) = 0.5528
  off(A') = 0.0859
4 step:
  off(A) = 0.0859
  off(A') = 0.0321
5 step:
  off(A) = 0.0321
  off(A') = 0.0001
6 step:
  off(A) = 0.0001
  off(A') = 0.0000
7 step:
  off(A) = 0.0000
  off(A') = 0.0000
Stopping computing...
```

```
Default Matrix:
  1.0000 -1.0000  0.0000
 -1.0000  0.0000  1.0000
  0.0000  1.0000  1.0000

Matrix of Eigenvalues:
  2.0000  0.0000 -0.0000
 -0.0000 -1.0000  0.0000
 -0.0000  0.0000  1.0000

Eigenvalues:
  2.0000 -1.0000  1.0000
Eigenvectors:
  0.5774  0.4082  0.7071
 -0.5774  0.8165 -0.0000
 -0.5774 -0.4082  0.7071

Verification:
Eigenvalues (numpy):
 -1.0000  1.0000  2.0000
Eigenvectors (numpy):
  0.4082 -0.7071 -0.5774
  0.8165  0.0000  0.5774
 -0.4082 -0.7071  0.5774
```

Тестовый пример 3

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

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

Computing eigenvalues...

1 step:

off(A) = 8.0000
off(A') = 6.0000

2 step:

off(A) = 6.0000
off(A') = 4.0000

3 step:

off(A) = 4.0000
off(A') = 2.0000

4 step:

off(A) = 2.0000
off(A') = -0.0000

Stopping computing...

Default Matrix:

1.0000	1.0000	1.0000	0.0000
1.0000	1.0000	0.0000	1.0000
1.0000	0.0000	1.0000	1.0000
0.0000	1.0000	1.0000	1.0000

Matrix of Eigenvalues:

3.0000	0.0000	0.0000	-0.0000
0.0000	1.0000	0.0000	0.0000
0.0000	-0.0000	1.0000	0.0000
-0.0000	0.0000	0.0000	-1.0000

Eigenvalues:

3.0000	1.0000	1.0000	-1.0000
--------	--------	--------	---------

Eigenvectors:

0.5000	-0.5000	-0.5000	0.5000
0.5000	0.5000	-0.5000	-0.5000
0.5000	-0.5000	0.5000	-0.5000
0.5000	0.5000	0.5000	0.5000

Verification:

Eigenvalues (numpy):

-1.0000	1.0000	3.0000	1.0000
---------	--------	--------	--------

Eigenvectors (numpy):

0.5000	-0.7071	-0.5000	0.0000
-0.5000	-0.0000	-0.5000	0.7071
-0.5000	0.0000	-0.5000	-0.7071
0.5000	0.7071	-0.5000	0.0000

Тестовый пример 4

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

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

Вычисление собственных значений и векторов + проверка на сходимость:

```
Computing eigenvalues...
1 step:
    off(A) = 16.0
    off(A') = 8.0
2 step:
    off(A) = 8.0
    off(A') = 0.0
Stopping computing...

Default Matrix:
 1.0000  1.4142  2.0000
 1.4142  3.0000  1.4142
 2.0000  1.4142  1.0000

Eigenvalues:
 5.0000  0.0000  0.0000
 0.0000  1.0000 -0.0000
 0.0000 -0.0000 -1.0000

Eigenvectors:
 0.5000 -0.5000 -0.7071
 0.7071  0.7071  0.0000
 0.5000 -0.5000  0.7071
```

Проверка:

```
Verification:
Eigenvalues (numpy):
 5.0000 -1.0000  1.0000
Eigenvectors (numpy):
 -0.5000 -0.7071  0.5000
 -0.7071 -0.0000 -0.7071
 -0.5000  0.7071  0.5000
```

Тестовый пример 5.

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

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

Matrix of Eigenvalues:

6.6458 -0.0000 0.0000 0.0000
0.0000 -3.6458 0.0000 0.0000
0.0000 0.0000 1.3542 0.0000
0.0000 0.0000 -0.0000 1.6458

Eigenvalues:

6.6458 -3.6457 1.3543 1.6458

Eigenvectors:

0.7258 -0.2209 -0.2706 -0.5926
0.2419 0.9658 -0.0902 -0.0226
0.5544 -0.0606 0.7479 0.3601
0.3276 -0.1212 -0.5994 0.7202

Verification:

Eigenvalues (numpy):

6.6458 -3.6458 1.6458 1.3542

Eigenvectors (numpy):

0.7258 0.2209 -0.5926 0.2706
0.2419 -0.9658 -0.0226 0.0902
0.5544 0.0606 0.3601 -0.7479
0.3276 0.1212 0.7202 0.5994

```
Computing eigenvalues..  
1 step:  
    off(A) = 24.0000  
    off(A') = 16.0000  
2 step:  
    off(A) = 16.0000  
    off(A') = 6.4107  
3 step:  
    off(A) = 6.4107  
    off(A') = 1.4416  
4 step:  
    off(A) = 1.4416  
    off(A') = 0.7520  
5 step:  
    off(A) = 0.7520  
    off(A') = 0.3023  
6 step:  
    off(A) = 0.3023  
    off(A') = 0.0526  
7 step:  
    off(A) = 0.0526  
    off(A') = 0.0116  
8 step:  
    off(A) = 0.0116  
    off(A') = 0.0004  
9 step:  
    off(A) = 0.0004  
    off(A') = 0.0002
```

```
10 step:  
    off(A) = 0.0002  
    off(A') = 0.0001  
11 step:  
    off(A) = 0.0001  
    off(A') = 0.0000  
12 step:  
    off(A) = 0.0000  
    off(A') = 0.0000  
13 step:  
    off(A) = 0.0000  
    off(A') = 0.0000  
Stopping computing...  
  
Default Matrix:  
4.0000 2.0000 2.0000 1.0000  
2.0000 -3.0000 1.0000 1.0000  
2.0000 1.0000 3.0000 1.0000  
1.0000 1.0000 1.0000 2.0000
```

Тестовый пример 6

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

$$A = \begin{bmatrix} 1 & -1 & -2 & 1 & 1 \\ -1 & 0 & 1 & 3 & 2 \\ -2 & 1 & 3 & 1 & 1 \\ 1 & 3 & 1 & 4 & 0 \\ 1 & 2 & 1 & 0 & 5 \end{bmatrix}$$

Computing eigenvalues..

1 step:

off(A) = 46.0000
off(A') = 28.0000

2 step:

off(A) = 28.0000
off(A') = 20.0000

3 step:

off(A) = 20.0000
off(A') = 13.7812

4 step:

off(A) = 13.7812
off(A') = 10.6958

5 step:

off(A) = 10.6958
off(A') = 8.4956

6 step:

off(A) = 8.4956
off(A') = 6.1340

7 step:

off(A) = 6.1340
off(A') = 4.0985

8 step:

off(A) = 4.0985
off(A') = 2.8706

9 step:

off(A) = 2.8706
off(A') = 1.7825

11 step:

off(A) = 1.0100
off(A') = 0.3413

12 step:

off(A) = 0.3413
off(A') = 0.1815

13 step:

off(A) = 0.1815
off(A') = 0.0735

14 step:

off(A) = 0.0735
off(A') = 0.0027

15 step:

off(A) = 0.0027
off(A') = 0.0013

16 step:

off(A) = 0.0013
off(A') = 0.0006

17 step:

off(A) = 0.0006
off(A') = 0.0002

18 step:

off(A) = 0.0002
off(A') = 0.0001

19 step:

off(A) = 0.0001
off(A') = 0.0000

20 step:

off(A) = 0.0000
off(A') = 0.0000

21 step:

off(A) = 0.0000
off(A') = 0.0000

22 step:

off(A) = 0.0000
off(A') = 0.0000

23 step:

off(A) = 0.0000
off(A') = 0.0000

24 step:

off(A) = 0.0000
off(A') = 0.0000

Stopping computing...

Default Matrix:

1.0000	-1.0000	-2.0000	1.0000	1.0000
-1.0000	0.0000	1.0000	3.0000	2.0000
-2.0000	1.0000	3.0000	1.0000	1.0000
1.0000	3.0000	1.0000	4.0000	0.0000
1.0000	2.0000	1.0000	0.0000	5.0000

Matrix of Eigenvalues:

-0.0346	-0.0000	0.0000	0.0000	-0.0000
-0.0000	-2.8049	-0.0000	0.0000	0.0000
0.0000	-0.0000	4.0174	-0.0000	0.0000
-0.0000	0.0000	-0.0000	7.1581	0.0000
-0.0000	0.0000	-0.0000	0.0000	4.6640

Eigenvalues:

-0.0345	-2.8050	4.0174	7.1581	4.6640
---------	---------	--------	--------	--------

Eigenvectors:

0.6446	0.4462	-0.5905	-0.0148	0.1910
-0.4889	0.7360	-0.0297	0.4500	-0.1263
0.5827	0.1432	0.6718	0.3885	-0.1940
0.0593	-0.4111	-0.4456	0.5458	-0.5753
-0.0496	-0.2641	-0.0223	0.5902	0.7609

Verification:

Eigenvalues (numpy):

-2.8049 -0.0346 4.0174 4.6640 7.1581

Eigenvectors (numpy):

0.4462	0.6446	0.5905	0.1910	-0.0148
0.7360	-0.4889	0.0297	-0.1263	0.4500
0.1433	0.5827	-0.6718	-0.1940	0.3885
-0.4111	0.0593	0.4456	-0.5753	0.5458
-0.2641	-0.0496	0.0223	0.7609	0.5902

Тестовый пример 7

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

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Computing eigenvalues...

1 step:

off(A) = 4.0000
off(A') = 2.0000

2 step:

off(A) = 2.0000
off(A') = 1.0000

3 step:

off(A) = 1.0000
off(A') = 0.2113

4 step:

off(A) = 0.2113
off(A') = 0.0580

5 step:

off(A) = 0.0580
off(A') = 0.0006

6 step:

off(A) = 0.0006
off(A') = 0.0000

7 step:

off(A) = 0.0000
off(A') = 0.0000

Stopping computing...

Default Matrix:

2.0000	-1.0000	0.0000
-1.0000	2.0000	-1.0000
0.0000	-1.0000	2.0000

Matrix of Eigenvalues:

3.4142	0.0000	0.0000
-0.0000	0.5858	-0.0000
0.0000	-0.0000	2.0000

Eigenvalues:

3.4142	0.5858	2.0000
--------	--------	--------

Eigenvectors:

0.5000	0.5000	-0.7071
-0.7071	0.7071	0.0000
0.5000	0.5000	0.7071

Verification:

Eigenvalues (numpy):

3.4142	2.0000	0.5858
--------	--------	--------

Eigenvectors (numpy):

-0.5000	-0.7071	0.5000
0.7071	0.0000	0.7071
-0.5000	0.7071	0.5000

4.

Решение задания
Вариант 10

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

$$A = \begin{bmatrix} 4.33 & 0.81 & 2.67 & 0.92 & -0.53 \\ 0.81 & 4.33 & 0.81 & 2.67 & 0.92 \\ 2.67 & 0.81 & 4.33 & 0.81 & 2.92 \\ 0.92 & 2.67 & 0.81 & 4.33 & -0.53 \\ -0.53 & 0.92 & 2.92 & -0.53 & 4.33 \end{bmatrix}$$

```

1 step:
  off(A) = 54.0142
  off(A') = 36.9614
2 step:
  off(A) = 36.9614
  off(A') = 22.7036
3 step:
  off(A) = 22.7036
  off(A') = 12.4636
4 step:
  off(A) = 12.4636
  off(A') = 8.7183
5 step:
  off(A) = 8.7183
  off(A') = 3.9293
6 step:
  off(A) = 3.9293
  off(A') = 2.9636
7 step:
  off(A) = 2.9636
  off(A') = 2.1975
8 step:
  off(A) = 2.1975
  off(A') = 1.6841
9 step:
  off(A) = 1.6841
  off(A') = 1.3190

```

```

10 step:
  off(A) = 1.3190
  off(A') = 0.9196
11 step:
  off(A) = 0.9196
  off(A') = 0.6016
12 step:
  off(A) = 0.6016
  off(A') = 0.3237
13 step:
  off(A) = 0.3237
  off(A') = 0.1077
14 step:
  off(A) = 0.1077
  off(A') = 0.0335
15 step:
  off(A) = 0.0335
  off(A') = 0.0154
16 step:
  off(A) = 0.0154
  off(A') = 0.0091
17 step:
  off(A) = 0.0091
  off(A') = 0.0028
18 step:
  off(A) = 0.0028
  off(A') = 0.0009

```

```

19 step:
  off(A) = 0.0009
  off(A') = 0.0005
20 step:
  off(A) = 0.0005
  off(A') = 0.0001
21 step:
  off(A) = 0.0001
  off(A') = 0.0000
22 step:
  off(A) = 0.0000
  off(A') = 0.0000
23 step:
  off(A) = 0.0000
  off(A') = 0.0000
24 step:
  off(A) = 0.0000
  off(A') = 0.0000
25 step:
  off(A) = 0.0000
  off(A') = 0.0000
26 step:
  off(A) = 0.0000
  off(A') = 0.0000
Stopping computing...

```

Default Matrix:

4.3300	0.8100	2.6700	0.9200	-0.5300
0.8100	4.3300	0.8100	2.6700	0.9200
2.6700	0.8100	4.3300	0.8100	2.9200
0.9200	2.6700	0.8100	4.3300	-0.5300
-0.5300	0.9200	2.9200	-0.5300	4.3300

Matrix of Eigenvalues:

4.6669	-0.0000	0.0000	0.0000	0.0000
-0.0000	9.1894	0.0000	0.0000	-0.0000
0.0000	0.0000	6.2374	-0.0000	-0.0000
0.0000	0.0000	-0.0000	1.6199	0.0000
-0.0000	-0.0000	-0.0000	0.0000	-0.0636

Eigenvalues:

4.6669	9.1895	6.2374	1.6199	-0.0636
--------	--------	--------	--------	---------

Eigenvectors:

0.7281	0.4317	-0.0663	-0.2822	0.4467
-0.4218	0.4481	-0.3961	-0.6472	-0.2132
0.1799	0.5850	0.3906	0.2587	-0.6371
-0.1623	0.3876	-0.5838	0.6590	0.2197
-0.4830	0.3470	0.5876	0.0147	0.5485

Verification:

Eigenvalues (numpy):

9.1894	-0.0636	1.6199	4.6669	6.2374
--------	---------	--------	--------	--------

Eigenvectors (numpy):

0.4317	0.4467	-0.2822	-0.7281	-0.0663
0.4481	-0.2132	-0.6472	0.4218	-0.3961
0.5850	-0.6371	0.2587	-0.1799	0.3906
0.3876	0.2197	0.6590	0.1623	-0.5838
0.3470	0.5485	0.0147	0.4830	0.5876

5.

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.

Доказана сходимость матрицы получаемой в результате преобразования подобия, а также не изменность нормы Фробениуса, что говорит о том, что сама “тяжесть” матрицы фактически не менялась, а просто “смешалась в центр”. Разобраны 7 тестовых примеров, которые говорят о том, что алгоритм корректно отрабатывает для любой симметричной матрицы, а также проведена сверка с пакетом для определения собственных векторов и значений.

6.

Программная реализация

```

def check_equal_dim(matrix):
    if matrix.shape[0] != matrix.shape[1]:
        return False
    return True

def frobenius_norm(matrix):
    if not check_equal_dim(matrix):
        raise ValueError("Matrix isn't n, n dimension")

    n = matrix.shape[0]
    f_norm = 0
    for i in range(n):
        for j in range(n):
            f_norm += abs(matrix[i, j])**2
    f_norm = np.sqrt(f_norm)
    return f_norm

def off(matrix):
    if not check_equal_dim(matrix):
        raise ValueError("Matrix isn't n, n dimension")

    n = matrix.shape[0]
    trace_sum = 0
    for i in range(n):
        trace_sum += matrix[i, i]**2

    off_res = frobenius_norm(matrix)**2 - trace_sum
    return off_res

def calc_non_diag(matrix):
    if not check_equal_dim(matrix):
        raise ValueError("Matrix isn't n, n dimension")

    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):
    if not check_equal_dim(matrix):
        raise ValueError("Matrix isn't n, n dimension")

    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

def make_zeros(matrix):
    if not check_equal_dim(matrix):
        raise ValueError("Matrix isn't n, n dimension")

    n = matrix.shape[0]
    for i in range(n):
        for j in range(n):
            if abs(matrix[i, j]) < 10**-10:
                matrix[i, j] = 0
    return matrix

```