МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

**Кафедра компьютерных технологий и систем**

# Лабораторная работа №2 по курсу «ВМА»
# «Проблема собственных значений»

Вариант 2

Выполнил
Статкевич Захар Дмитриевич
студент 3-го курса 6 группы

Преподаватель:
Будник А. М.

Минск, 2023

# ОГЛАВЛЕНИЕ

# Входные данные.

Дана матрица:

$$\begin{bmatrix} 0.6444 & 0 & -0.1683 & 0.1184 & 0.1973 \\ -0.0395 & 0.4208 & 0 & -0.0802 & 0.0263 \\ 0.0132 & -0.1184 & 0.7627 & 0.0145 & 0.046 \\ 0.0395 & 0 & -0.096 & 0.7627 & 0 \\ 0.0263 & -0.0395 & 0.1907 & -0.0158 & 0.5523 \end{bmatrix}$$

Домножив на $A^T$ слева, получим симметричную матрицу $A^T A$:

$$\begin{bmatrix} 0.41923779 & -0.01922333 & -0.09716147 & 0.10936737 & 0.14123396 \\ -0.01922333 & 0.19265145 & -0.09783633 & -0.03484086 & -0.01619521 \\ -0.09716147 & -0.09783633 & 0.65561867 & -0.08509983 & 0.10720222 \\ 0.10936737 & -0.03484086 & -0.08509983 & 0.60262178 & 0.01319172 \\ 0.14123396 & -0.01619521 & 0.10720222 & 0.01319172 & 0.34677027 \end{bmatrix}$$

В дальнейшем под матрицей $A$ понимаем матрицу $A^\top A$.

# Глава 1

# Метод Крылова

## 1.1 Постановка задачи.

Нам необходимо:

1. найти коэффиценты собственного многочлена;
2. найти сообственные значения;
3. найти сообственные вектора;
4. вычислить вектор невязки, оценить его норму;
5. вычислить невязку коэффицентов собственного многочлена;

## 1.2 Алгоритм.

Берем произвольный вектор $c^0$, размерность которого согласована с матрицей $A$, и по этому вектору составим рекуррентную последовательность векторов:

$$c^0, \quad c^1 = Ac^0, \quad c^2 = Ac^1 = A^2c^0, \quad \dots$$

Записываем систему:

$$q_1c^{n-1} + \cdots + q_nc^0 = c^n$$

Решаем систему, находим коэффициенты $q_1, \dots, q_n$ - коэффициенты собственного многочлена $P_n(\lambda)$.
Находим корни $P_n(\lambda)$, получаем последовательность собственных значений $\lambda_1, \dots, \lambda_n$

Для каждого $\lambda_i$ находим коэффициенты $\beta_{1i}, \ldots, \beta_{n,i}$:

$$\begin{cases} \beta_{i1} = 1 \\ \beta_{i2} = \lambda_i - q_1 \\ \beta_{i3} = {\lambda_i}^2 - q_1\lambda_i - q_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \beta_{in} = {\lambda_i}^{n-1} - q_1{\lambda_i}^{n-2} - \cdots - q_{n-1} \end{cases}$$

Тогда собственный вектор $x_i$ соответствующий собственному значению $\lambda_i$ будем искать в виде:

$$x_i = \beta_{i1}c^{n-1} + \beta_{i2}c^{n-2} + \cdots + \beta_{in}c^0$$

***Примечание***

В качестве вектора $c^0$ возьмем вектор $(1, 0, \ldots, 0)^\top$

## 1.3 Реализация.

```
def krylov(A, n):
    a = np.array(A)
    At = a.transpose()
    a = np.dot(At, a)

    c = []
    c.append([1, 0, 0, 0, 0])
    for i in range(1, n + 1):
        c.append(np.dot(a, c[i - 1]))
    for el in c:
        print("\t", el)

    C = np.array(c)
    cn = c.pop()
    c = np.array(c).transpose()
    for i in range(n):
        c[i] = list(reversed(c[i]))
    p = np.linalg.solve(c, cn)

    x = Symbol('x')
    Lambda = solve(x**5 - p[0] * x**4 - p[1] * x**3 - p[2] * x**2 - p[3] * x -
       p[4], x)

    l = max(Lambda)

    b = np.ones(n)
    for i in range(1, n):
```

```
        b[i] = b[i - 1] * l - p[i - 1]
    x = np.sum([b[i] * C[n - i - 1] for i in range(n)], axis=0)
    r = np.dot(a, x) - l * x

    rnorm = np.linalg.norm(r, 1)
    p = np.insert(p, 0, -1)
    r1 = sum(-(l ** (n - i)) * p[i] for i in range(n + 1))
```

## 1.4 Выводы по полученным результатам.

Коэффициенты собственного многочлена $P(\lambda)$:

$$[2.21689996\ -1.82259145\ 0.68304846\ -0.11469321\ 0.0069546]$$

Собственные значения $\lambda$:

$$[0.1528,\ 0.2053,\ 0.4574,\ 0.6205,\ 0.7808]$$

Максимальное собственное $\lambda_{\text{Max}}$:

$$0.780834944860558$$

Коэффициенты $B^i$:

$$[1\ -1.43606502\ 0.7012617\ -0.13547881\ 0.00890662]$$

Собственный вектор матрицы $A - x(\lambda_{\text{Max}})$:

$$[0.00217131\ 0.00053496\ -0.0048239\ 0.00350193\ -0.00039841]$$

Вектор невязки $r$:

$$[2.2877 \times 10^{-16},\ -2.4937 \times 10^{-18},\ 0,\ 2.2551 \times 10^{-17},\ 7.8605 \times 10^{-18}]$$

Норма невязки $||r||$:

$$2.61672194329376 \times 10^{-16}$$

Норма невязки коэффицентов многочлена:

$$1.30971622436249 \times 10^{-16}$$

## Вывод.

С помощью метода Крылова мы нашли все коэффиценты собственного многочлена. Для найденного собственного вектора $x$ матрицы  соответствующего максимальному собственному значению норма вектора невязки получилась порядка $10^{-16}$, что говорит о том, что собственный вектор найден с достаточной точностью.

# Глава 2

# Метод вращений

## 2.1 Постановка задачи.

Нам необходимо:

1. найти собственные значения с заданной точностью;
2. найти собственные вектора;
3. вычислить вектор невязки, оценить его норму;

## 2.2 Алгоритм.

По заданной матрице $A$ будем строить последовательность матриц $A^k$ так, что каждая следующая матрица $A^{k+1}$ получается из матрицы $A^k$ при помощи преобразования подобия с матрицей вращения $T_{ij}$:

$$T_{ij} = \begin{array}{c} \\ \\ i \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & i & & j & & \end{array} \\ \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \cos\varphi & \dots & -\sin\varphi & \vdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & \sin\varphi & \dots & \cos\varphi & \vdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \end{array}$$

Предположим, что преобразования доведенные до шага $k$ и построена матрица $A^k = \left\{a_{ij}^k\right\}$. Найдем в ней наибольший по модулю не диагональный элемент $a_{ij}^k, i \neq j$. Поскольку матрица $A$ симметрическая, то искать элемент можно в верхнем треугольнике $i < j$ (если таких элементов несколько, то возьмем любой из них, в качестве максимального по модулю элемента). По индексам $i, j$ строим матрицу $T_{ij}^k = T_{ij}\left(\varphi^k\right)$.

Значение $\sin\varphi$ и $\cos\varphi$ находим по формулам:

$$\operatorname{tg} 2\varphi^k = \frac{2a_{ij}^k}{a_{ii}^k - a_{jj}^k}, \quad \left|\varphi^k\right| \leqslant \frac{\pi}{4}$$

$$\cos 2\varphi^k = \frac{1}{\sqrt{1 + \operatorname{tg}^2 2\varphi^k}}.$$

$$\cos\varphi^k = \sqrt{\frac{1 + \cos 2\varphi^k}{2}}$$

$$\sin\varphi^k = \pm\sqrt{\frac{1 - \cos 2\varphi^k}{2}}$$

Знак в формуле для sin выбирается тот же, что и у выражения:

$$a_{ij}^k \left(a_{ii}^k - a_{jj}^k\right)$$

Тогда строим последовательность:

$$A^{k+1} = \left(T_{ij}^k\right)^T A^k T_{ij}^k, \quad k = 0, 1, \ldots, \quad A^0 = A$$

И:

$$U^{k+1} = U^k \cdot T_{ij}^k, \quad k = 0, 1, \ldots, \quad U^0 = E$$

$$A^0 = A, A^1, A^2, \ldots, A^k, \ldots \longrightarrow \Lambda$$

$$U^0 = E, U^1, U^2, \ldots, U^k, \ldots \longrightarrow U$$

Вычисления продолжаем до тех пор, пока не выполнится условие:

$$\left|t\left(A^{k+1}\right)\right| \leqslant \varepsilon$$

где $t(A)$ - мера близости, определяемая выше.
В итоге в рамках заданной точности $\varepsilon$ диагональные элементы последней матрицы $A^{k+1}$ принимаются за искомые собственные значения:

$$\lambda_i = a_{ii}^{k+1}, \quad i = \overline{1, n}$$

Столбцы матрицы $U^{k+1}$ принимаются за координаты собственных векторов, соответствующих этим собственным значениям.

***Примечание:*** Априорное количество итераций можем найти по формуле:

$$k_{apr} \geq \frac{\lg \varepsilon - \lg t(A)}{\lg q}$$

Где $q$ :

$$q = 1 - \frac{2}{n(n-1)}$$

## 2.3 Реализация.

```
epsilon = 10 ** (-5)
def sum(matrix):
    mask = np.eye(matrix.shape[0], dtype=bool)
    masked_matrix = np.ma.masked_array(matrix, mask)
    sum_of_squares = np.sum(np.square(masked_matrix))
    if sum_of_squares < 0:
        sum_of_squares = - sum_of_squares
    print(f"{sum_of_squares} <= {epsilon}")
    return sum_of_squares
def max(matrix):
    mask = np.eye(matrix.shape[0], dtype=bool)
    masked_matrix = np.ma.masked_array(matrix, mask)
    absolute_values = np.abs(masked_matrix)
    max_index = np.argmax(absolute_values)
    return max_index
def rotation_method(matrix):
    len = matrix.shape[0]
    counter = 0
    A = matrix.dot(matrix.transpose())
    #A = matrix
    eigenvectors = np.eye(len)
    while (sum(A) > epsilon):
        print(f"\n---------------iteration {counter}------------------\n")
        print(f"\n{A}\n")
        max_index_row, max_index_col = np.unravel_index(max(A), A.shape)
        tg2 = (2 * A[max_index_row][max_index_col]) / (A[max_index_row][
           max_index_row] - A[max_index_col][max_index_col])
        print(f"tg:{tg2}")
        cos2 = (1 + tg2 ** 2) ** (-0.5)
        cos = ((1 + cos2) / 2) ** 0.5
        sin = ((1 - cos2) / 2) ** 0.5
        if tg2 < 0:
            sin = -sin

        T = np.eye(len)
        T[max_index_row][max_index_row] = cos
        T[max_index_col][max_index_col] = cos
        T[max_index_col][max_index_row] = sin
        T[max_index_row][max_index_col] = -sin
```

```
        T_transpose = T.transpose()
        print(f"\n{T}\n")
        A = T_transpose.dot(A)
        A = A.dot(T)
        print(f"\n{A}\n")
        counter +=1

        A[max_index_row][max_index_col] = 0
        A[max_index_col][max_index_row] = 0
        eigenvectors = eigenvectors.dot(T)

    print(f"\n-------------- Result ------------------\n")
    matrix = matrix.dot(matrix.transpose())

    for i in range(matrix.shape[0]):
        test_matrix = np.eye(len).dot(A[i][i])
        result = matrix.dot(eigenvectors[:, i]) - test_matrix.dot(eigenvectors
            [:, i])
    return eigenvectors
```

## 2.4 Выводы по полученным результатам.

Собственное значение: 0.4574,

$$[0.6578\ 0.1022\ -0.1998\ -0.6030\ 0.3915]$$

Собственное значение: 0.1528,

$$[0.2625\ 0.8122\ 0.3229\ 0.0540\ -0.4053]$$

Собственное значение: 0.7808,

$$[-0.4519\ 0.0270\ 0.6530\ -0.5713\ 0.2056]$$

Собственное значение: 0.6205,

$$[0.4017\ -0.1712\ 0.5876\ 0.5087\ 0.4530]$$

Собственное значение: 0.2053,

$$[-0.3643\ 0.5476\ -0.2902\ 0.2198\ 0.6595]$$

Норма невязки максимального по модулю собственного значения :

$$6.743301558763751 \times 10^{-5}$$

Количество итераций: 15.

## Вывод.

С помощью метода вращений мы нашли все собственные значения и соответствующий им спектр с точностью порядка $10^{-16}$ за 15 итераций для эпсилона порядка $10^{-5}$. Невязка собственного вектора максимального собственного значения также довльно близка к нулю (порядка $10^{-16}$), что означает, что собственное значение также найдено правильно.

# Глава 3

# Метод Данилевского

## 3.1 Постановка задачи.

Нам необходимо:

1. найти сообственные значения;
2. найти сообственные вектора;
3. вычислить вектор невязки, оценить его норму;

## 3.2 Алгоритм.

Согласно методу Данилевского нам нужно найти такую матрицу $S$, что $\Phi = S^{-1}AS$

То есть метода Данилевского состоит в том, чтобы последовательными преобразованиями матрицы $A$ сделать ее вида Фробениуса.

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

Где коэффициенты $p_1, p_2, \dots, p_{n-1}, p_n$ - коэффициенты многочлена $P_n(\lambda)$.

***В лабораторной работе представлен регулярный случай и все дальнейшие формулы, описывают алгоритм регулярного случая.***

Алгоритм:

$$A_1 = M_{n-1}^{-1} A M_{n-1}$$

$$A_2 = M_{n-2}^{-1} A_1 M_{n-2}$$

И так далее, после выполнения $n-1$-го шага преобразования мы получим матрицу:

$$A_{n-1} = \underbrace{M_1^{-1}M_2^{-1}\dots M_{n-1}^{-1}}_{S^{-1}} A \underbrace{M_{n-1}\dots M_2 M_1}_{S} = \begin{pmatrix} a_{11}^{n-1} & a_{12}^{n-1} & \dots & a_{1n-1}^{n-1} & a_{1n}^{n-1} \\ 1 & 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & \vdots & 1 & 0 \end{pmatrix} =$$

$$= \begin{bmatrix} p_1 & p_2 & \dots & p_{n-1} & p_n \\ 1 & 0 & \dots & 0 & 0 \\ \dots & \ddots & \dots & \dots & \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} = \Phi.$$

Где:

$$M_{n-1} = \begin{pmatrix} 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ -\frac{a_{n1}}{a_{nn-1}} & \dots & \frac{1}{a_{nn-1}} & -\frac{a_{nn}}{a_{nn-1}} \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

*Матрица* $M_{n-1}^{-1}$ *существует, так как мы выбираем* $|M_{n-1}| = \frac{1}{a_{nn-1}} \neq 0$

$$M_{nn-1}^{-1} = \begin{pmatrix} 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \dots & a_{nn-1} & a_{nn} \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Для того, чтобы найти собственный вектор исходной матрицы $A$ нужно домножить на $S$ :

$$x = Sy = M_{n-1}\dots M_1 y.$$

Здесь в качестве собственного вектора мы можем выбрать

$$y = \left(\lambda_i^{n-1}, \lambda_i^{n-2}\dots, \lambda_i, 1\right),$$

При умножении вектора $y$ на $M_i$ будет изменяться только одна координата этого вектора. Такой способ построения собственных векторов годится только для регулярного случая.

## 3.3 Реализация.

```
    import numpy as np
from sympy import symbols, solve

A = np.array([[0.6444, 0.0000, -0.1683, 0.1184, 0.1973],
              [-0.0395, 0.4208, 0.0000, -0.0802, 0.0263],
              [0.0132, -0.1184, 0.7627, 0.0145, 0.0460],
              [0.0395, 0.0000, -0.0960, 0.7627, 0.0000],
              [0.0263, -0.0395, 0.1907, -0.0158, 0.5523]])

At = A.transpose()

a = np.dot(At, A)

n = len(A)
f = a
s = np.identity(n)

for i in range(n - 1):
    m = np.identity(n)
    m[n - 2 - i] = f[n - 1 - i]

    f = np.dot(m, f)
    f = np.dot(f, np.linalg.inv(m))
    s = np.dot(s, np.linalg.inv(m))

p = f[0]

x = symbols('x')
Lambda = solve(x**5 - p[0] * x**4 - p[1] * x**3 - p[2] * x**2 - p[3] * x - p
    [4], x)

maxLambda = max(Lambda)
print("\nmax(lambda):\n\t", maxLambda)
for i in Lambda:
    y = [i ** j for j in range(n - 1, -1, -1)]
    y = np.dot(s, y)
    r = np.dot(a, y) - maxLambda * y
```

## 3.4 Выводы по полученным результатам.

Коэффициенты собственного многочлена $P(\lambda)$:

$$[2.216,\ -1.823,\ 0.683,\ -0.115,\ 0.007]$$

Собственные значения $\lambda$:

$$[0.153,\ 0.205,\ 0.457,\ 0.621,\ 0.781]$$

Максимальное значение $\lambda_{\max}$:

$$0.781$$

Собственный вектор матрицы:

$$[-0.961,\ -2.344,\ -0.879,\ -0.144,\ 1.000]$$

Собственный вектор матрицы:

$$[-0.783,\ 0.807,\ -0.192,\ 0.212,\ 1.000]$$

Собственный вектор матрицы:

$$[1.204,\ 0.150,\ -0.386,\ -1.188,\ 1.000]$$

Собственный вектор матрицы:

$$[0.867,\ -0.452,\ 1.186,\ 1.278,\ 1.000]$$

Собственный вектор матрицы:

$$[-5.450,\ -1.343,\ 12.108,\ -8.790,\ 1.000]$$

Норма невязки максимального по модулю собственного значения:

$$2.052^{-15}$$

Норма невязки коэффицентов многочлена: $4.423^{-12}$

## Вывод.

С помощью метода Данилевского мы нашли коэффициенты собственного многочлена, а так же систему собственных векторов. Для найденного собственного вектора x матрицы A соответствующего максимальному собственному значению норма вектора невязки получилась порядка $10^{-15}$, что говорит о том, что собственный вектор найден с достаточной точностью. Стоит отметить меньшую точность, согласно нормы вектора невязки, по сравнению с методом Крылова.

# Глава 4

# Степенной метод

## 4.1 Постановка задачи.

Нам необходимо:

1. найти максимальное сообственное значение;
2. найти соответствующий ему сообственный вектор;
3. вычислить вектор невязки, оценить его норму;

## 4.2 Алгоритм.

Степенной метод является итерационным методом решения полной (теоретически, а на практике частичной) проблемы собственных значений.

Суть метода заключается в последовательном приближении $y^k$ к собственному вектору соответствующему максимальному собственному значению $\lambda$. За $\lambda^k$ берётся отношение соответствующий произвольных координат векторов $y^{k+1}$ и $y^k$. Итерационный процесс останавливается, когда $\left|\lambda^{k+1} - \lambda^k\right| \leq \varepsilon$.

Возьмём начальное приближение $y^0 = (1, 0, \ldots, 0)$, а последующее будем вычислять как:

$$y^{k+1} = Ay^k$$

$$\lambda^k \approx \frac{y_0^{k+1}}{y_0^k}$$

за $x$ можно принять $x \approx y^{k+1}$.

## 4.3 Реализация.

```
epsilon = 10 ** (-5)
def power_method(matrix):
    counter = 0
    A = matrix.dot(matrix.transpose())
    # A = matrix
```

```
    A_power = A
    y_0 = np.array([1, 0, 0, 0, 0]).transpose()
    z_k = y_0

    lambda_1 = np.dot(A.dot(z_k), z_k) / np.dot(z_k, z_k)
    lambda_ = lambda_1 - 2 * epsilon

    while abs(lambda_1 - lambda_) > epsilon:
        print(f"{abs(lambda_1 - lambda_)} <= {epsilon}")
        print(f"\n------------------iteration {counter}------------------\n")
        print("z:")
        A_power = A_power.dot(A)
        norm = np.linalg.norm(A_power.dot(y_0), ord=1)
        z_k = A_power.dot(y_0)
        print(z_k)
        print(f"norm:{norm}")
        z_k = z_k.dot((norm) ** (-1))
        print(z_k)
        lambda_ = lambda_1
        print("lambda:")
        print(lambda_)
        lambda_1 = np.dot(A.dot(z_k), z_k) / np.dot(z_k, z_k)
        print(f"{lambda_1}\n")
        counter += 1

    print(f"{abs(lambda_1 - lambda_)} <= {epsilon}")
    print(f"\n-------------- Result ------------------\n")
    matrix = matrix.dot(matrix.transpose())
    test_matrix = np.eye(5).dot(lambda_)
    print(matrix.dot(z_k).transpose())
    print(test_matrix.dot(z_k).transpose())
    matrix = matrix.dot(z_k)
    test_matrix = test_matrix.dot(z_k)
    result = matrix - test_matrix
    print(np.linalg.norm(result.transpose(), 1))
    return A
```

## 4.4 Выводы по полученным результатам.

Максимальное собственное значение: 0.7808

Соответствующий ему собственный вектор

$$[0.2386,\ -0.0149,\ -0.3393,\ 0.3016,\ -0.1056]$$

Невязка $r$:

$$[-0.0003,\ 0.0001,\ -0.0004,\ -0.0004,\ -0.0003]$$

Норма вектора невязки: 0.001598

Количество итераций: 18

## Вывод.

С помощью степенного метода мы нашли максимальное по модулю собственное значение и соответствующий ему собственный с точностью порядка $10^{-4}$ за 18 итераций для эпсилона порядка $10^{-5}$. Собственное значение и собственный вектор также совпадают с полученными ранее методами Крылова и Данилевского, правда с меньшей точностью.

# Глава 5

# Сравнительный анализ

| | Сложность | Память | Норма невязки $r_{\lambda max}$ | Количество итераций |
|---|---|---|---|---|
| Метод Крылова | $O(n^3)$ | $O(n^2)$ | $2.61 \times 10^{-16}$ | |
| Метод вращений | $O(kn^2)$ | $O(n^2)$ | $6.74 \times 10^{-5}$ | 15 |
| Метод Данилевского | $O(n^3)$ | $O(n^2)$ | $2.05 \times 10^{-15}$ | |
| Степенной метод | $O(kn^2)$ | $O(n^2)$ | $1.59 \times 10^{-4}$ | 18 |

Сравнивая с методом Данилевского, метод Крылова является более точным (разница в невязках на порядок), однако и более трудоёмким. Данные методы удобно ипользовать, когда мы решаем полную ПЗС.

Сравнивая итерационные методы: метод вращений и степенной метод, можем наблюдать, что метод вращений позволил найти решение немного быстрее, чем метод степенной метод (15 итераций против 18). А так же сравнивая нормы векторов невязки, можем удтверждать, что найденное решение методом вращений имеет лучшую точность. Делаем вывод, что точность (норма вектора невязки) напрямую зависит от количества итераций. Так же стоит отметить, что степенной метод удобен при решении частичной ПЗС.