**Московский авиационный институт (национальный исследовательский университет)**

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

**Лабораторная работа по предмету "Численные методы" №1**

Студент: Федоров М.А.

Преподаватель: Ревизников Д.Л.

Группа: М8О-307Б-22

Дата:

Оценка:

Подпись:

Оглавление

[**Задача 1.1** 3](#_Toc192860597)

[**Общий алгоритм решения** 3](#_Toc192860598)

[**Программное решение** 3](#_Toc192860599)

[**Пример работы** 5](#_Toc192860600)

[**Задача 1.2** 7](#_Toc192860601)

[**Общий алгоритм решения** 7](#_Toc192860602)

[**Программное решение** 7](#_Toc192860603)

[**Пример работы для системы из 5-и уравнений** 8](#_Toc192860604)

[**Задача 1.3** 8](#_Toc192860605)

[**Общий алгоритм решения метода итераций** 9](#_Toc192860606)

[**Программное решение** 9](#_Toc192860607)

[**Пример работы** 11](#_Toc192860608)

[**Общий алгоритм решения метода Зейделя** 11](#_Toc192860609)

[**Программное решение** 12](#_Toc192860610)

[**Пример работы** 13](#_Toc192860611)

[**Задача 1.4** 13](#_Toc192860612)

[**Общий алгоритм решения** 14](#_Toc192860613)

[**Программное решение** 15](#_Toc192860614)

[**Пример работы** 16](#_Toc192860615)

[**Задача 1.5** 17](#_Toc192860616)

[**Общий алгоритм решения** 17](#_Toc192860617)

[**Программное решение** 18](#_Toc192860618)

[**Пример работы** 20](#_Toc192860619)

[**Вывод** 21](#_Toc192860620)

# **Задача 1.1**

Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

# **Общий алгоритм решения**

Необходимо вычислить решение для следующей системы:



Для вычисления разложения LU = A программа для каждого столбца матрицы A обнуляет поддиагональные элементы, выбирая сам диагональный элемент в качестве ведущего и прибавляя строку с ведущим элементом, домноженную на некоторый коэффициент, к строкам ниже. Коэффициенты домножения записываются в матрицу L, а по окончании обнуления поддиагональных элементов исходной матрицы A мы получаем матрицу U. Для получения вектора решений x решаются 2 системы: Lz=b и Ux=z. Определитель матрицы A можно вычислить, перемножив все диагональные элементы U. Для вычисления обратной матрицы A^-1 в начале работы программы берется единичная матрица E, с которой выполняются те же операции, что и при вычислении матрицы U. После чего для каждого столбца E решается система Ux=столбец E. Из полученных решений x состоит обратная матрица A^-1.

# **Программное решение**

def LU():

    A = np.matrix([[-7., -2., -1., -4., -12.],

         [-4., 6., 0., -4., 22.],

         [-8., 2., -9., -3., 51.],

         [0., 0., -7., 1., 49.]])

    U = A[:, :-1].copy()

    b = A[:, -1:].copy()# вектор свободных членов

    b = b.transpose()

    L = np.eye(4, 4)

    E = np.eye(4, 4) # вспомогательная матрица для поиска обратной

    p = 0 #кол-во перестановок

    # поиск  LU разложения

    for i in range (0, 3):

        print("Итерация ", i+1, "\nВыберите номер ведущего элемента столбца", i+1, ":")

        for j in range(i, 4):

            print(j+1, ": ", U[j, i], end="\n")

        while (True):

            elem = int(input())-1

            if elem > 4 or elem < i:

                print("Неправильный номер элемента")

            elif (U[elem, i] == 0):

                print("Нельзя выбрать нулевой элемент")

            else:

                print("Выбран элемент ", U[elem, i], end="\n")

                if elem == i:

                    break

                else:

                    buffer = np.copy(U[i])

                    U[i] = np.copy(U[elem])

                    U[elem] = np.copy(buffer)

                    p += 1

                    buffer = np.copy(L[i, :i])

                    L[i, :i] = np.copy(L[elem, :i])

                    L[elem, :i] = np.copy(buffer)

                    buffer = np.copy(b[0, i])

                    b[0, i] = np.copy(b[0, elem])

                    b[0, elem] = np.copy(buffer)

                    buffer = np.copy(E[i])

                    E[i] = np.copy(E[elem])

                    E[elem] = np.copy(buffer)

                    break

        for j in range(i+1, 4):

            mu\_j = (U[j, i]/U[i, i])

            #print(mu\_j)

            L[j, i] = mu\_j

            #print(string, end="\n\n")

            U[j] = U[j]-U[i]\*mu\_j

            E[j] = E[j]-E[i]\*mu\_j

        print("Верхняя матрица:\n", U, "\nНижняя матрица:\n", L, end="\n\n")

    z = np.zeros([1, 4])

    x = np.zeros([1, 4])

    # решение Lz=b

    for i in range(0, 4):

        summ = 0

        for j in range(0, i):

            summ += L[i, j]\*z[0, j]

        z[0, i] = b[0, i]-summ

    # Решение Ux=z

    for i in range(3, -1, -1):

        summ = 0

        for j in range(i+1, 4):

            summ += U[i, j]\*x[0, j]

        x[0, i] = (1 / U[i, i])\*(z[0, i]-summ)

    print("Решения СЛАУ в векторе x: ", x)

    det = np.pow(-1, p)

    for i in range(0, 4):

        det \*= U[i, i]

    # print(, "Определитель: ", det)

    print("Определитель матрицы A: ", det, "\nПроверка определителя: ", np.linalg.det(A[:, :-1]))

    # поиск обратной матрицы

    A\_reverse = np.zeros([4, 4])

    for i in range(0, 4):

        A\_reverse[3, i] = E[3, i] / U[3, 3]

        for j in range(2, -1, -1):

            summ = 0

            for k in range(3, j, -1):

                summ += A\_reverse[k, i] \* U[j, k]

            A\_reverse[j, i] = (E[j, i] - summ) / U[j, j]

    print("Обратная матрица A^(-1):\n", A\_reverse, "\nПроверка обратной матрицы:\n", np.linalg.inv(A[:, :-1]))

    #проверка правильности решения с помощью вычисления вектора b

    print("Проверка правильности решения (Свободные члены, вычесленные из вектора x и свободные члены вектора b):")

    for i in range (0, 4):

        answer = 0.

        for j in range(0, 4):

            answer += A[i, j]\*x[0, j]

        print(answer, "\t\t", A[i, -1])

# **Пример работы**

Итерация 1

Выберите номер ведущего элемента столбца 1 :

1 : -7.0

2 : -4.0

3 : -8.0

4 : 0.0

3

Выбран элемент -8.0

Верхняя матрица:

[[-8. 2. -9. -3. ]

[ 0. 5. 4.5 -2.5 ]

[ 0. -3.75 6.875 -1.375]

[ 0. 0. -7. 1. ]]

Нижняя матрица:

[[ 1. 0. 0. 0. ]

[ 0.5 1. 0. 0. ]

[ 0.875 0. 1. 0. ]

[-0. 0. 0. 1. ]]

Итерация 2

Выберите номер ведущего элемента столбца 2 :

2 : 5.0

3 : -3.75

4 : 0.0

2

Выбран элемент 5.0

Верхняя матрица:

[[-8. 2. -9. -3. ]

[ 0. 5. 4.5 -2.5 ]

[ 0. 0. 10.25 -3.25]

[ 0. 0. -7. 1. ]]

Нижняя матрица:

[[ 1. 0. 0. 0. ]

[ 0.5 1. 0. 0. ]

[ 0.875 -0.75 1. 0. ]

[-0. 0. 0. 1. ]]

Итерация 3

Выберите номер ведущего элемента столбца 3 :

3 : 10.25

4 : -7.0

4

Выбран элемент -7.0

Верхняя матрица:

[[-8. 2. -9. -3. ]

[ 0. 5. 4.5 -2.5 ]

[ 0. 0. -7. 1. ]

[ 0. 0. 0. -1.78571429]]

Нижняя матрица:

[[ 1. 0. 0. 0. ]

[ 0.5 1. 0. 0. ]

[-0. 0. 1. 0. ]

[ 0.875 -0.75 -1.46428571 1. ]]

Решения СЛАУ в векторе x: [[ 6. 3. -8. -7.]]

Определитель матрицы A: -500.0

Проверка определителя: -499.99999999999983

Обратная матрица A^(-1):

[[ 0.248 0.236 -0.46 0.556]

[-0.208 0.044 0.16 -0.176]

[-0.08 -0.06 0.1 -0.26 ]

[-0.56 -0.42 0.7 -0.82 ]]

Проверка обратной матрицы:

[[ 0.248 0.236 -0.46 0.556]

[-0.208 0.044 0.16 -0.176]

[-0.08 -0.06 0.1 -0.26 ]

[-0.56 -0.42 0.7 -0.82 ]]

Проверка правильности решения (Свободные члены, вычесленные из вектора x и свободные члены вектора b):

-12.000000000000004 -12.0

22.000000000000004 22.0

51.0 51.0

49.0 49.0

# **Задача 1.2**

Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.

# **Общий алгоритм решения**

Метод прогонки решает СЛАУ для трехдиагональной матрицы. В таком случае каждое уравнение системы записывается как **ax+bx+cx=d**. Программа получает на вход все значения коэффициентов a, b, c, d в качестве векторов. Для решения СЛАУ составляются вспомогательные вектора P и Q, элементы которых вычисляются по формулам:  
Изображение выглядит как Шрифт, линия, число, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Затем можно вычислить решения x по следующим формулам:

Изображение выглядит как текст, Шрифт, рукописный текст, белый

Контент, сгенерированный ИИ, может содержать ошибки.

# **Программное решение**

def running\_method(n: int):

    #a = np.array([0., -9., 5., -6., 2.])

    #b = np.array([-11., 17., 20., -20., 8.])

    #c = np.array([9., 6., 8., 7., 0.])

    #d = np.array([-117., -97., -6., 59., -86.])

    a = np.empty(n)

    b = np.empty(n)

    c = np.empty(n)

    d = np.empty(n)

    print("Вводите последовательно на каждой строке элементы a, b, c и свободный член (на первой строке a = 0, на последней c = 0), разделяя элементы пробелом")

    for i in range(0, n):

        a[i], b[i], c[i], d[i] = [float(x) for x in input().split()]

    a[0] = 0

    c[-1] = 0

    print("Вектор a:", a, "\nВектор b:", b, "\nВектор c:", c, "\nВектор d (свободные члены):", d, end="\n\n")

    P = np.zeros([1, n])

    Q = np.zeros([1, n])

    x = np.zeros([1, n])

    P = P[0]

    Q = Q[0]

    x = x[0]

    P[0] = (-1\*c[0]) / b[0]

    Q[0] = d[0] / b[0]

    for i in range(1, n):

        P[i] = (-1\*c[i]) / (b[i]+(a[i]\*P[i-1]))

        Q[i] = (d[i]-(a[i]\*Q[i-1])) / (b[i]+(a[i]\*P[i-1]))

    x[n-1] = Q[n-1]

    for i in range(n-2, -1, -1):

        x[i] = P[i]\*x[i+1]+Q[i]

    #print(P)

    #print(Q)

    print("Вектор решений (x):", x, end="\n")

# **Пример работы для системы из 5-и уравнений**

Вводите последовательно на каждой строке элементы a, b, c и свободный член (на первой строке a = 0, на последней c = 0), разделяя элементы пробелом

0 -11 9 -117

-9 17 6 -97

5 20 8 -6

-6 -20 7 59

2 8 0 -86

Вектор a: [ 0. -9. 5. -6. 2.]

Вектор b: [-11. 17. 20. -20. 8.]

Вектор c: [9. 6. 8. 7. 0.]

Вектор d (свободные члены): [-117. -97. -6. 59. -86.]

Вектор решений (x): [ 9. -2. 3. -7. -9.]

# **Задача 1.3**

Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

# **Общий алгоритм решения метода итераций**

СЛАУ приводится к эквивалентному виду:  
Изображение выглядит как текст, Шрифт, рукописный текст, белый

Контент, сгенерированный ИИ, может содержать ошибки.

Или же в матрично-векторной форме:  
Изображение выглядит как текст, рукописный текст, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Компоненты β и α эквивалентной системы вычисляются по формулам:  


Решение методом итераций имеет вид:  
Изображение выглядит как Шрифт, текст, рукописный текст, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Где k – конечная итерация. Критерием окончания служит неравенство:  


Где ε – заданная точность. Причем количество итераций для достижения заданной точности можно вычислить по формуле:  
Изображение выглядит как текст, Шрифт, линия, типография

Контент, сгенерированный ИИ, может содержать ошибки.

# **Программное решение**

def simple\_iterations(n, accuracy: float):

'''

A = np.matrix([[-25., 4., -4., 9.],

[-9., 21., 5., -6.],

[9., 2., 19., -7.],

[-7., 4., -7., 25.]])

b = np.array([86., 29., 28., 68.])

A = np.matrix([[10., 1., 1.],

[2., 10., 1.],

[2., 2., 10.]])

b = np.array([12., 13., 14.])

'''

A = np.zeros([n, n])

b = np.zeros(n)

print("Вводите последовательно на каждой строке элементы элементы матрицы A и свободный член, разделяя элементы пробелом")

for i in range(0, n):

print("Элементы уравнения на", i+1, "строке:")

inpt = input().split()

for j in range(0, n):

A[i, j] = float(inpt[j])

b[i] = inpt[-1]

for i in range(0, n):

if A[i, i] == 0:

for j in range(i+1, n):

if A[j, i] != 0:

buffer = np.copy(A[i, i])

A[i, i] = np.copy(A[j, i])

A[j, i] = np.copy(buffer)

buffer = np.copy(b[i])

b[i] = np.copy(b[j])

b[j] = np.copy(buffer)

break

b[i] = b[i] / A[i, i]

for j in range(0, n):

if j != i:

A[i, j] = -1\*A[i, j] / A[i, i]

A[i, i] = 0

x = np.copy(b)

x = x.reshape(-1, 1) # транспонирование вектора (делаем его вертикальным)

b = b.reshape(-1, 1) # транспонирование вектора (делаем его вертикальным)

x\_prev = np.copy(x)

x = b + np.dot(A, x)

# критерий окончания: норма разности x с итераций i и i-1 не должна превышать accuracy

while np.linalg.norm(x - x\_prev) > accuracy: # linalg.norm по-умолчанию использует Норму Фробениуса или 2-норма (корень суммы квадратов всех элементов матрицы/вектора)

x\_prev = np.copy(x)

x = b + np.dot(A, x)

print("Норма разности x с последней итерации:", np.linalg.norm(x - x\_prev), "\nВектор решений x:\n", x, end="\n")

# **Пример работы**

Вводите последовательно на каждой строке элементы элементы матрицы A и свободный член, разделяя элементы пробелом

Элементы уравнения на 1 строке:

-25 4 -4 9 86

Элементы уравнения на 2 строке:

-9 21 5 -6 29

Элементы уравнения на 3 строке:

9 2 19 -7 28

Элементы уравнения на 4 строке:

-7 4 -7 25 68

Норма разности x с последней итерации: 0.00791464173631252

Вектор решений x:

[[-3.00128059e+00]

[-5.91756500e-04]

[ 3.99794899e+00]

[ 3.00148737e+00]]

# **Общий алгоритм решения метода Зейделя**

СЛАУ приводится к эквивалентному виду:  
Изображение выглядит как текст, Шрифт, линия, рукописный текст

Контент, сгенерированный ИИ, может содержать ошибки.

Где k – номер итерации. В матрично-векторном виде система выглядит так:  


Если сравнить данный вид решения с методом простых итераций, то можно заметить, что α = B+C, где В - нижняя треугольная матрица с диагональными элементами , равными нулю, а С - верхняя треугольная матрица с диагональными элементами, отличными от нуля. Следовательно:



Откуда:



Таким образом, метод Зейделя является методом простых итераций с матрицей правых α =(E-B)-1C частей и вектором правых частей (E-B)-1β и, следовательно, сходимость и погрешность метода Зейделя можно исследовать с помощью формул, выведенных для метода простых итераций, в которых вместо матрицы α подставлена матрица (E-B)-1C, а вместо вектора правых частей – вектор (E-B)-1β. Для практических вычислений важно, что в качестве достаточных условий сходимости метода Зейделя могут быть использованы условия, приведенные выше для метода простых итераций.

# **Программное решение**

def zeydel(n, accuracy: float):

    '''

    A = np.matrix([[-25., 4., -4., 9.],

         [-9., 21., 5., -6.],

         [9., 2., 19., -7.],

         [-7., 4., -7., 25.]])

    b = np.array([86., 29., 28., 68.])

    A = np.matrix([[10., 1., 1.],

         [2., 10., 1.],

         [2., 2., 10.]])

    b = np.array([12., 13., 14.])

    '''

    size = n

    B = np.zeros([size, size]) # нижняя треугольная с нулевой диагональю

    C = np.zeros([size, size]) # верхняя треугольная с ненулевой диагональю

    E = np.eye(size)

    A = np.zeros([n, n])

    b = np.zeros(n)

    print("Вводите последовательно на каждой строке элементы элементы матрицы A и свободный член, разделяя элементы пробелом")

    for i in range(0, n):

        print("Элементы уравнения на", i+1, "строке:")

        inpt = input().split()

        for j in range(0, n):

            A[i, j] = float(inpt[j])

        b[i] = inpt[-1]

    for i in range(0, size):

        if A[i, i] == 0:

            for j in range(i+1, size):

                if A[j, i] != 0:

                    buffer = np.copy(A[i, i])

                    A[i, i] = np.copy(A[j, i])

                    A[j, i] = np.copy(buffer)

                    buffer = np.copy(b[i])

                    b[i] = np.copy(b[j])

                    b[j] = np.copy(buffer)

                    break

        b[i] = b[i] / A[i, i]

        for j in range(0, size):

            if j != i:

                A[i, j] = -1\*A[i, j] / A[i, i]

            if j < i:

                B[i, j] = np.copy(A[i, j])

            else:

                C[i, j] = np.copy(A[i, j])

        A[i, i] = 0

        C[i, i] = 0

    E\_B\_inv = np.linalg.inv(E-B)

    x = np.copy(b)

    x = x.reshape(-1, 1) # транспонирование вектора (делаем его вертикальным)

    b = b.reshape(-1, 1) # транспонирование вектора (делаем его вертикальным)

    x\_prev = np.copy(x)

    x = np.dot(E\_B\_inv, np.dot(C, x\_prev)) + np.dot(E\_B\_inv, b)

    # критерий окончания: норма разности x с итераций i и i-1 не должна превышать accuracy

    while np.linalg.norm(x - x\_prev) > accuracy: # linalg.norm по-умолчанию использует Норму Фробениуса или 2-норма (корень суммы квадратов всех элементов матрицы/вектора)

        x\_prev = np.copy(x)

        x = np.dot(E\_B\_inv, np.dot(C, x\_prev)) + np.dot(E\_B\_inv, b)

    print("Норма разности x с последней итерации:", np.linalg.norm(x - x\_prev), "\nВектор решений x:\n", x, end="\n")

# **Пример работы**

Вводите последовательно на каждой строке элементы элементы матрицы A и свободный член, разделяя элементы пробелом

Элементы уравнения на 1 строке:

-25 4 -4 9 86

Элементы уравнения на 2 строке:

-9 21 5 -6 29

Элементы уравнения на 3 строке:

9 2 19 -7 28

Элементы уравнения на 4 строке:

-7 4 -7 25 68

Норма разности x с последней итерации: 0.0028465859658694794

Вектор решений x:

[[-2.99941563e+00]

[ 5.88732408e-04]

[ 3.99958957e+00]

[ 2.99995451e+00]]

# **Задача 1.4**

Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

# **Общий алгоритм решения**

Метод применим только для симметричных матриц. Пусть известна матрица А(k) на k-й итерации (для k=0 A(0)=A). Тогда алгоритм решения следующий:

1. Выбирается максимальный по модулю недиагональный элемент aij(k) матрицы A(k):



1. Ставится задача найти такую ортогональную матрицу U(k), чтобы в результате преобразования подобия A(k+1)=U(k)TA(k)U(k) произошло обнуление элемента aij(k+1) матрицы A(k+1). В качестве ортогональной матрицы выбирается матрица вращения, имеющая следующий вид:

Изображение выглядит как текст, диаграмма, линия, число

Контент, сгенерированный ИИ, может содержать ошибки.

Где ϕ вычисляется по формуле:



Причем, если aii(k)= ajj(k), то:



1. Строится матрица A(k+1)=U(k)TA(k)U(k)

В качестве критерия окончания выбирается условие малости суммы квадратов внедиагональных элементов:



В качестве искомых собственных значений принимаются: 

Координатными столбцами собственных векторов матрицы A в единичном базисе будут столбцы матрицы U, т е:



# **Программное решение**

def rotation\_method(n, accuracy: float):

    '''

    A = np.matrix([[-8., -4., 8.],

         [-4., -3., 9.],

         [8., 9., -5.]])

    A = np.matrix([[4., 2., 1.],

         [2., 5., 3.],

         [1., 3., 6.]])

    '''

    A = np.zeros([n, n])

    print("Вводите последовательно на каждой строке элементы элементы матрицы A, разделяя их пробелом")

    for i in range(0, n):

        print("Элементы матрицы на", i+1, "строке:")

        inpt = input().split()

        for j in range(0, n):

            A[i, j] = float(inpt[j])

    A\_starting = np.copy(A)

    own\_vectors = np.eye(3)

    iterations = 0

    while(True):

        iterations += 1

        U = np.eye(3)

        criterion = 0

        ii = 0

        jj = 1

        phi = 0

        for i in range(0, 3):

            for j in range(i+1, 3):

                if np.abs(A[i, j]) > np.abs(A[ii, jj]):

                    ii = i

                    jj = j

        # print(A[ii, jj], ii, jj)

        if np.abs(A[ii, ii] - A[jj, jj]) < 0.001:

            phi = np.pi / 4

        else:

            phi = 0.5 \* np.arctan(2\*A[ii, jj] / (A[ii, ii] - A[jj, jj]))

        # print(phi, np.sin(phi), np.cos(phi))

        U[ii, ii] = np.cos(phi)

        U[ii, jj] = -1\*np.sin(phi)

        U[jj, ii] = np.sin(phi)

        U[jj, jj] = np.cos(phi)

        A = np.dot(np.dot(np.transpose(U), A), U)

        own\_vectors = np.dot(own\_vectors, U)

        for i in range(0, 3):

            for j in range(i+1, 3):

                criterion += np.pow(A[i, j], 2)

                # print(criterion, A[i, j])

        criterion = np.pow(criterion, 0.5)

        # print("Критерий:", criterion, "\nA:\n", A, "\nU:\n", U, end="\n\n\n")

        if (criterion <= accuracy):

            print("Собственные векторы (транспонированные):")

            for i in range(0, 3):

                print("СВ", i, " = ", own\_vectors[:, i])

            print("Собственные значения:")

            for i in range(0, 3):

                print("СЗ", i, " = ", A[i, i])

            print("Проверка на ортогональность собственных векторов:")

            for i in range(0, 3-1):

                for j in range(i+1, 3):

                    print("(x", i, ", x", j, ") = ", np.dot(own\_vectors[:, i], np.transpose(own\_vectors[:, j])))

            print("Проверка на выполнение равенста A\*СВ=СЗ\*СВ:")

            for i in range(0, 3):

                print(np.dot(A\_starting, np.transpose(own\_vectors[:, i])), " = ", A[i, i] \* np.transpose(own\_vectors[:, i]))

            print("Кол-во итераций:", iterations, "\nПогрешность:", criterion)

            break

# **Пример работы**

Вводите последовательно на каждой строке элементы элементы матрицы A, разделяя их пробелом

Элементы матрицы на 1 строке:

-8 -4 8

Элементы матрицы на 2 строке:

-4 -3 9

Элементы матрицы на 3 строке:

8 9 -5

Собственные векторы (транспонированные):

СВ 0 = [ 0.76123518 -0.58796387 0.27353153]

СВ 1 = [0.23772224 0.64546044 0.7258574 ]

СВ 2 = [-0.6033317 -0.48752365 0.63111928]

Собственные значения:

СЗ 0 = -2.0297343916096

СЗ 1 = 5.630239684410162

СЗ 2 = -19.60050529280056

Проверка на ортогональность собственных векторов:

(x 0 , x 1 ) = 4.600524716746792e-17

(x 0 , x 2 ) = 5.733362751762236e-17

(x 1 , x 2 ) = -2.4732585664973036e-17

Проверка на выполнение равенста A\*СВ=СЗ\*СВ:

[-1.54977371 1.18073466 -0.56945104] = [-1.54510522 1.19341048 -0.55519635]

[1.3232395 3.64544629 4.08163488] = [1.33843319 3.63409697 4.08675112]

[ 11.82570246 9.55597127 -12.3699629 ] = [ 11.82560623 9.55570998 -12.37025674]

Кол-во итераций: 5

Погрешность: 0.019642587754060113

# **Задача 1.5**

Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

# **Общий алгоритм решения**

Для реализации разложения A=QR используется матрица Хаусхолдера:



Где v1 – вектор-столбец матрицы A.

Положим A0=A. Построим преобразование Хаусхолдера A1=H1A0:  
Изображение выглядит как диаграмма, линия, Шрифт, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Вектор v определяется следующим образом:

Изображение выглядит как Шрифт, текст, рукописный текст, белый

Контент, сгенерированный ИИ, может содержать ошибки.

Матрица Хаусхолдера вычисляется согласно формуле:

Изображение выглядит как Шрифт, часы, дизайн, типография

Контент, сгенерированный ИИ, может содержать ошибки.

На следующей итерации строим преобразование A2=H2A1, в котором v2 вычисляется следующим образом:Изображение выглядит как Шрифт, текст, рукописный текст, белый

Контент, сгенерированный ИИ, может содержать ошибки.

Повторяя процесс n-1 раз получим:  


Для вычисления собственных значений строится итерационный процесс:  
Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

При отсутствии у матрицы кратных собственных значений последовательность A(k) сходится к верхней треугольной матрице (в случае, когда все собственые значения вещественные) или к верхней квазитреугольной матрице (если имеются комплексно-сопряженные пары собственных значений). Для каждого вещественного собственного его поддиагональные элементы будут стремиться к нулю. Для таких элементов критерием окончания можно считать:  


А собственное значение принимается равным диагональному элементу данного столбца. Если же имеется поддиагональный элемент, не стремящийся к нулю, то этот элемент входит в блок 2х2, соответствующий комплексно-сопряженной паре собственных значений, которые можно найти, решив уравнение:  


Где ajj(k) – верхний левый элемент блока 2х2, соответствующего комплексно-сопряженной паре. Для таких блоков используется критерий окончания, отличный от случая с вещественными собственными значениями:  


# **Программное решение**

def QR(n, accuracy: float):

'''

A = np.matrix([[-3., 1., -1.],

[6., 9., -4.],

[5., -4., -8.]])

A = np.matrix([[1., 3., 1.],

[1., 1., 4.],

[4., 3., 1.]])

'''

A = np.zeros([n, n])

print("Вводите последовательно на каждой строке элементы элементы матрицы A, разделяя их пробелом")

for i in range(0, n):

print("Элементы матрицы на", i+1, "строке:")

inpt = input().split()

for j in range(0, n):

A[i, j] = float(inpt[j])

complex\_list = list()

A\_original = np.copy(A)

E = np.eye(3)

first\_iteration = True

k = 0

while(True):

A\_previous = np.copy(A)

Q = np.eye(3)

v = np.zeros([3, 1])

for i in range(0, 3-1):

v\_i = np.copy(v)

for j in range(0, i):

v\_i[j, 0] = 0

v\_i[i, 0] = A[i, i]+np.sign(A[i, i])\*np.pow(sum([np.pow(A[j, i], 2) for j in range(i, 3)]), 0.5)

for j in range(i+1, 3):

v\_i[j, 0] = A[j, i]

H = E - 2\*(np.dot(v\_i, np.transpose(v\_i)) / np.dot(np.transpose(v\_i), v\_i))

A = np.dot(H, A)

Q = np.dot(Q, H)

# print(A, end="\n\n")

# print(Q, "\n\n", np.linalg.inv(Q), "\n\n", np.transpose(Q))

# print(A, "\n\n", A\_original, "\n\n", np.dot(Q, A))

R = np.copy(A)

print("Итерация", k, ":")

A = np.dot(R, Q)

print(Q, "\n\n", R, "\n\n", A, end="\n\n\n")

float\_criterion = 0

complex\_criterion = 0

for i in range(0, 3):

#проверка на комплексные СЗ

if i+1 < 3:

if (not first\_iteration and np.abs(A[i+1, i]) > np.abs(A\_previous[i+1, i])\*2 and [i+1, i] not in complex\_list):

complex\_list.append([i+1, i])

print("Комплексное СЗ по элементу", i+1, i)

elif [i+1, i] not in complex\_list:

float\_criterion += np.pow(A[i+1, i], 2)

for j in range(i+2, 3):

float\_criterion += np.pow(A[j, i], 2)

float\_criterion = np.pow(float\_criterion, 0.5)

#Вычисление разности вещественных частей комплексных СЗ с разных итераций для критерия окончания. В каждом элементе complex\_list хранится индекс [i+1, i]

for i in complex\_list:

complex\_criterion += np.abs((A\_previous[i[0]-1, i[1]] + A\_previous[i[0], i[1]+1]) - (A[i[0]-1, i[1]] + A[i[0], i[1]+1])) / 2

if float\_criterion <= accuracy and complex\_criterion <= accuracy:

print("Вещественные собственные значения матрицы:")

for i in range(0, 3):

if ([i+1, i] not in complex\_list) and ([i, i-1] not in complex\_list):

print(A[i, i])

print("Комплексные собственные значения матрицы:")

#В каждом элементе complex\_list хранится индекс [i+1, i]

for i in complex\_list:

# решаем уравнение вида ax^2+bx+c=0 где x - пара комплексных СЗ; коэффициенты a, b, c вычисляются ниже

# a = 1

# b = -1(A[ii]+A[jj])

# c = (A[ii]\*A[jj] - A[ij]\*A[ji])

# где j = i+1

b = -1\*(A[i[0]-1, i[1]] + A[i[0], i[1]+1])

c = A[i[0]-1, i[1]]\*A[i[0], i[1]+1] - A[i[0]-1, i[1]+1]\*A[i[0], i[1]]

D = np.pow(b, 2) - 4 \* c

x1 = (-1\*b + np.lib.scimath.sqrt(D)) / 2

x2 = (-1\*b - np.lib.scimath.sqrt(D)) / 2

print(x1)

print(x2)

break

k += 1

first\_iteration = False

# **Пример работы**

Вводите последовательно на каждой строке элементы элементы матрицы A, разделяя их пробелом

Элементы матрицы на 1 строке:

-3 1 -1

Элементы матрицы на 2 строке:

6 9 -4

Элементы матрицы на 3 строке:

5 -4 -8

Итерация 0 :

[[-4.7 -9.61786996 4.72147135]

[-8.01641063 4.71949483 -1.86702043]

[-2.80891753 -3.1817719 -2.01949483]]

Итерация 1 :

[[-6.04092769 6.76848783 -3.84514099]

[ 7.87916432 7.15616351 5.22402774]

[ 0.97381767 -0.82113853 -3.11523582]]

Итерация 2 :

[[-4.27701055 -8.11521768 7.44102389]

[-7.76788242 5.94355746 -0.32850683]

[-0.36141082 -0.3655312 -3.66654691]]

…

Итерация 33 :

[[ 1.03137633e+01 3.98078728e-02 3.81011195e+00]

[ 9.35328995e-03 -7.86145770e+00 6.81777965e+00]

[ 3.11862230e-12 -7.22247523e-09 -4.45230559e+00]]

Вещественные собственные значения матрицы:

10.31376329041873

-7.861457696942279

-4.4523055934764715

Комплексные собственные значения матрицы:

# **Вывод**

В ходе выполнения лабораторной работы я научился применять алгоритмы, находящие решения СЛАУ, собственные значения и собственные вектора матриц.