

Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 1
по курсу «Численные методы»
Тема: численные методы решения СЛАУ.

Студент: Сыроежкин К.Г.
Группа: 80-304б
Преподаватель: Гидаспов В.Ю.
Оценка:

Москва, 2021

Задание 1

1) Постановка задачи: 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.

Вариант 16:

$$\begin{cases} -5 \cdot x_1 - x_2 - 3 \cdot x_3 - x_4 = 18 \\ -2 \cdot x_1 + 8 \cdot x_2 - 4 \cdot x_4 = -12 \\ -7 \cdot x_1 - 2 \cdot x_2 + 2 \cdot x_3 - 2 \cdot x_4 = 6 \\ 2 \cdot x_1 - 4 \cdot x_2 - 4 \cdot x_3 + 4 \cdot x_4 = -12 \end{cases}$$

2) Теория:

LU – разложение матрицы A представляет собой разложение матрицы A в произведение нижней и верхней треугольных матриц, т.е. $A = LU$

где L - нижняя треугольная матрица, U- верхняя треугольная матрица.

Чтобы получить LU разложение, необходимо выполнить прямой ход метода Гаусса. Рассмотрим k-ый шаг метода, на котором осуществляется обнуление поддиагональных элементов k-го столбца матрицы A. С этой целью используется следующая операция:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \mu_i^{(k)} a_{kj}^{(k-1)}, \quad \mu_i^{(k)} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \quad i = \overline{k+1, n}, \quad j = \overline{k, n}.$$

Такая операция эквивалентна умножению: $A^{(k)} = M_k A^{(k-1)}$, причем M имеет вид

$$M_k^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\mu_{k+1}^{(k)} & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & -\mu_n^{(k)} & 0 & 0 & 1 \end{pmatrix} \quad M_k^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \mu_{k+1}^{(k)} & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \mu_n^{(k)} & 0 & 0 & 1 \end{pmatrix}$$

В результате прямого хода метода Гаусса получим $A^{(n-1)} = U$, $L = M_1^{-1} M_2^{-1} \dots M_{n-1}^{-1}$. Таким образом, искомое разложение $A = LU$ получено.

В дальнейшем LU - разложение может быть эффективно использовано при решении систем алгебраических уравнений вида $Ax = b$. Действительно,

подставляя LU - разложение в СЛАУ, получим $LUx = b$, или $Ux = L^{-1}b$.
Т.е процесс решения СЛАУ сводится к двум простым этапам.

На первом этапе решается СЛАУ $Lz = b$. Поскольку матрица системы - нижняя треугольная, решение можно записать в явном виде

$$z_1 = b_1, \quad z_i = b_i - \sum_{j=1}^{i-1} l_{ij} z_j, \quad i = \overline{2, n}.$$

На втором этапе решается СЛАУ $Ux = z$ с верхней треугольной матрицей. Здесь, как и на предыдущем этапе, решение представляется в явном виде:

$$x_n = \frac{z_n}{u_{nn}}, \quad x_i = \frac{1}{u_{ii}} \left(z_i - \sum_{j=i+1}^n u_{ij} x_j \right), \quad i = \overline{n-1, 1}.$$

3) Полученный ответ:

```

A
[[-5 -1 -3 -1]
 [-2  0  8 -4]
 [-7 -2  2 -2]
 [ 2 -4 -4  4]]

B
[[ 18]
 [-12]
 [ 6]
 [-12]]

L
[[ 1.         0.         0.         0.         ]
 [-0.28571429  1.         0.         0.         ]
 [ 0.28571429 -0.125      1.         0.         ]
 [ 0.71428571 -0.09375    -0.67857143  1.         ]]

U
[[-7.         -2.         2.         -2.         ]
 [ 0.         -4.57142857 -3.42857143  3.42857143 ]
 [ 0.         0.         7.         -3.         ]
 [ 0.         0.         0.         -1.28571429 ]]

x
[[-2.]
 [ 3.]
 [-3.]
 [-2.]]

Проверка корней AX = B:
[[ 18.]
 [-12.]

```

```
[ 6.]
[-12.]]
Обратная матрица:
[[ 0.22222222  0.22222222 -0.33333333  0.11111111]
 [-0.33333333 -0.33333333  0.25         -0.29166667]
 [-0.33333333 -0.08333333  0.25         -0.04166667]
 [-0.77777778 -0.52777778  0.66666667 -0.13888889]]
Проверка обратной матрицы  $AA^{-1} = E$ :
[[ 1.00000000e+00  3.33066907e-16  2.22044605e-15 -1.11022302e-16]
 [-3.66373598e-15  1.00000000e+00 -7.99360578e-15  2.77555756e-16]
 [ 1.38777878e-17 -2.77555756e-17  1.00000000e+00  1.38777878e-16]
 [-4.44089210e-16 -1.11022302e-16 -1.77635684e-15  1.00000000e+00]]
Определитель:
-288.0
```

4) Код программы:

```
import numpy as np
def get_mat(file):
    mat = []
    answ = []
    with open(file) as file_handler:
        for line in file_handler:
            row = list(map(int, line.split(',')[ :-1]))
            mat.append(row[ :-1])
            answ.append([row[ -1]])
    mat = np.asarray(mat)
    answ = np.asarray(answ).reshape(-1,1)
    return np.matrix(mat), np.matrix(answ)

def get_LU(mat):
    lu_mat = np.matrix(np.zeros([mat.shape[0],mat.shape[1]]))
    n = mat.shape[0]
    for k in range(n):
        for j in range(k, mat.shape[1]):
            lu_mat[k, j] = mat[k, j] - lu_mat[k, :k] * lu_mat[:k, j]
        for i in range(k+1, n):
            lu_mat[i,k] = (mat[i,k] - lu_mat[i, :k] * lu_mat[:k, k]) / lu_mat[k,
k]
    return np.matrix(lu_mat)

def get_L(lu_mat):
    L = lu_mat.copy()
    for i in range(L.shape[0]):
        L[i, i] = 1
        L[i, i+1:] = 0
    return np.matrix(L)
```

```

def get_U(lu_mat):
    U = lu_mat.copy()
    for i in range(1, U.shape[0]):
        U[i, :i] = 0
    return np.matrix(U)

def LU_decomposition(mat):
    mat = np.matrix(mat)
    lu = get_LU(mat)
    matL = get_L(lu)
    matU = get_U(lu)
    return L, U

def solve(lu_mat,b):

    y = np.matrix(np.zeros([lu_mat.shape[0], 1]))
    for i in range(y.shape[0]):
        y[i, 0] = b[i, 0] - lu_mat[i, :i] * y[:i]
    x = np.matrix(np.zeros([lu_mat.shape[0], 1]))
    for i in range(1, x.shape[0] + 1):
        x[-i, 0] = (y[-i] - lu_mat[-i, -i:] * x[-i:, 0]) / lu_mat[-i, -i]
    return x

def invers(LU):
    E = np.matrix(np.eye(len(LU)))
    new_LU = LU.copy()
    for i in range(len(E)):
        new_LU[:,i] = solve(LU,E[:, i])
    return(new_LU)

def is_zero(mat,b):
    if np.allclose(b, np.zeros([b.shape[0],1])):
        b = mat[:, mat.shape[1]-1]
        mat = np.delete(mat, mat.shape[1]-1, 1)
    return(mat,b)

def det(U):
    determ = 1
    for i in range(U.shape[0]):
        determ *= U[i,i]

    return determ

A,answ = get_mat("test1.txt")
print("A\n",A)
print("B\n", answ)

```

```

L,U = LU_decomposition(A.tolist())
print("L\n",L)
print("U\n",U)
lu = get_LU(A)
print("x\n",solve(lu,answ))
print("Проверка корней AX = B:\n", np.dot(A,solve(lu,answ)))
print("Обратная матрица:\n", invers(lu))
print("Проверка обратной матрицы AA^-1 = E:\n",np.dot(invers(lu),A))
print("Определитель:\n", det(U))

```

5) Выводы: Метод Гаусса применяется как для аналитического решения СЛАУ, так и для нахождения обратной матрицы. Он позволяет получить наиболее точное решение и является менее трудоемким методом для матриц ограниченного размера.

Задание 2

1) Постановка задачи: 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.

Вариант 16:

$$\begin{cases} 18 \cdot x_1 - 9 \cdot x_2 = -81 \\ 2 \cdot x_1 - 9 \cdot x_2 - 4 \cdot x_3 = 71 \\ -9 \cdot x_2 + 21 \cdot x_3 - 8 \cdot x_4 = -39 \\ -4 \cdot x_3 - 10 \cdot x_4 + 5 \cdot x_5 = 64 \\ 7 \cdot x_4 + 12 \cdot x_5 = 3 \end{cases}$$

2) Теория:

Метод прогонки является одним из эффективных методов решения СЛАУ с трехдиагональными матрицами, возникающих при конечно-разностной аппроксимации задач для обыкновенных дифференциальных уравнений (ОДУ) и уравнений в частных производных второго порядка и является частным случаем метода Гаусса. Рассмотрим следующую СЛАУ:

$$a_1 = 0 \left\{ \begin{array}{l} b_1 x_1 + c_1 x_2 = d_1 \\ a_2 x_1 + b_2 x_2 + c_2 x_3 = d_2 \\ \quad a_3 x_2 + b_3 x_3 + c_3 x_4 = d_3 \\ \dots\dots\dots \\ a_{n-1} x_{n-2} + b_{n-1} x_{n-1} + c_{n-1} x_n = d_{n-1} \\ \quad a_n x_{n-1} + b_n x_n = d_n, \quad c_n = 0, \end{array} \right.$$

решение которой будем искать в виде:

$$x_i = P_i x_{i+1} + Q_i, \quad i = \overline{1, n}$$

где $P_i, Q_i, i = \overline{1, n}$ - прогоночные коэффициенты, подлежащие определению. Для их определения выразим из первого уравнения СЛАУ x_1 через x_2 , получим:

$$x_1 = \frac{-c_1}{b_1} x_2 + \frac{d_1}{b_1} = P_1 x_2 + Q_1, \quad P_1 = \frac{-c_1}{b_1}, \quad Q_1 = \frac{d_1}{b_1}.$$

$$x_2 = \frac{-c_2}{b_2 + a_2 P_1} x_3 + \frac{d_2 - a_2 Q_1}{b_2 + a_2 P_1} = P_2 x_3 + Q_2,$$

$$P_2 = \frac{-c_2}{b_2 + a_2 P_1}, \quad Q_2 = \frac{d_2 - a_2 Q_1}{b_2 + a_2 P_1}.$$

$$x_i = \frac{-c_i}{b_i + a_i P_{i-1}} x_{i+1} + \frac{d_i - a_i Q_{i-1}}{b_i + a_i P_{i-1}},$$

$$P_n = 0 \text{ (т.к. } c_n = 0), \quad Q_n = \frac{d_n - a_n Q_{n-1}}{b_n + a_n P_{n-1}} = x_n.$$

Обратный ход метода прогонки осуществляется в соответствии с выражением:

$$\left\{ \begin{array}{l} x_n = P_n x_{n+1} + Q_n = 0 \cdot x_{n+1} + Q_n = Q_n \\ x_{n-1} = P_{n-1} x_n + Q_{n-1} \\ x_{n-1} = P_{n-2} x_{n-1} + Q_{n-2} \\ \dots\dots\dots \\ x_1 = P_1 x_2 + Q_1. \end{array} \right. \quad \begin{array}{l} a_i \neq 0, \quad c_i \neq 0, \quad i = \overline{2, n-1} \\ |b_i| \geq |a_i| + |c_i|, \quad i = \overline{1, n}, \end{array}$$

A

```
[[ 18  -9   0   0   0]
 [  2  -9  -4   0   0]
 [  0  -9  21  -8   0]
 [  0   0  -4 -10   5]
 [  0   0   0   7  12]]
```

B

```
[[ -81]
 [  71]
 [ -39]
 [  64]
 [   3]]
```

X

```
[-8. -7. -6. -3.  2.]
```

Проверка $AX = B$:

```
[-81.  71. -39.  64.   3.]
```

4) Код программы:

```
def get_mat(file):
    mat = []
    answ = []
    with open(file) as file_handler:
        for line in file_handler:
            row = list(map(int, line.split(',')[::-1]))
            mat.append(row[::-1])
            answ.append([row[-1]])
    mat = np.asarray(mat)
    answ = np.asarray(answ).reshape(-1,1)
    return mat, answ

def get_PQ(mat,answ):
    P1 = -mat[0,1]/mat[0,0]
    Q1 = answ[0,0]/mat[0,0]
    size_mat = mat[:,0].size
    i = 1
    P = [P1]
    Q = [Q1]
    while i < size_mat-1:
        P.append(-mat[i,i+1]/(mat[i,i]+mat[i,i-1]*P[i-1]))
        Q.append((answ[i,0]-mat[i,i-1]*Q[i-1])/(mat[i,i]+mat[i,i-1]*P[i-1]))
        i+=1
```



```

P.append(0)
Q.append((answ[i,0]-mat[i,i-1]*Q[i-1])/(mat[i,i]+mat[i,i-1]*P[i-1]))
return P, Q

def method(size_mat, P, Q):
    X = np.zeros(size_mat)
    X[size_mat-1] = Q[size_mat-1]
    i = size_mat-2
    while i > -1:
        X[i] = X[i+1]*P[i]+Q[i]
        i-=1
    return X

def run(mat, answ):
    size_mat = mat[:,0].size
    answ = np.asarray(answ).reshape(-1,1)
    P, Q = get_PQ(mat,answ)
    X = method(size_mat, P, Q)
    return X

mat, answ = get_mat("test.txt")
mat = np.asarray(mat)
answ = np.asarray(answ).reshape(-1,1)
P, Q = get_PQ(mat,answ)
size_mat = mat[:,0].size
X = method(size_mat, P, Q)
print("A\n", mat)
print("B\n", answ)
print("X\n", X)
print("Проверка AX = B:\n", np.dot(mat,X))

```

5) Выводы:

Метод прогонки отлично подходит для решения СЛАУ с трехдиагональной матрицей, так как был разработан исключительно для этого. Является итерационным методом.

Задание 3

1) Постановка задачи:

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.

Вариант 16:

$$\begin{cases} 21 \cdot x_1 - 6 \cdot x_2 - 9 \cdot x_3 - 4 \cdot x_4 = 127 \\ -6 \cdot x_1 + 20 \cdot x_2 - 4 \cdot x_3 + 2 \cdot x_4 = -144 \\ -2 \cdot x_1 - 7 \cdot x_2 - 20 \cdot x_3 + 3 \cdot x_4 = 236 \\ 4 \cdot x_1 + 9 \cdot x_2 + 6 \cdot x_3 + 24 \cdot x_4 = -5 \end{cases}$$

2) Теория:

Методы последовательных приближений, в которых при вычислении последующего приближения решения используются предыдущие, уже известные приближенные решения, называются итерационными.

Рассмотрим СЛАУ:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

Приведем СЛАУ к эквивалентному виду:

$$\begin{cases} x_1 = \beta_1 + \alpha_{11}x_1 + \alpha_{12}x_2 + \dots + \alpha_{1n}x_n \\ x_2 = \beta_2 + \alpha_{21}x_1 + \alpha_{22}x_2 + \dots + \alpha_{2n}x_n \\ \dots \\ x_n = \beta_n + \alpha_{n1}x_1 + \alpha_{n2}x_2 + \dots + \alpha_{nn}x_n \end{cases}$$

В векторно-матричной форме:

$$x = \beta + \alpha x.$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \alpha = \begin{pmatrix} \alpha_{11} & \dots & \alpha_{1n} \\ \vdots & \dots & \vdots \\ \alpha_{n1} & \dots & \alpha_{nn} \end{pmatrix}.$$

Тогда метод простых итераций имеет вид:

$$\begin{cases} x^{(0)} = \beta \\ x^{(1)} = \beta + \alpha x^{(0)} \\ x^{(2)} = \beta + \alpha x^{(1)} \\ \dots \\ x^{(k)} = \beta + \alpha x^{(k-1)}. \end{cases}$$

Достаточное условие сходимости (преобладание диагональных элементов):

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}| \quad \forall i$$

Метод простых итераций довольно медленно сходится. Для его ускорения существует метод Зейделя

метод Зейделя для известного вектора x на k -ой итерации имеет вид:

$$\begin{cases} x_1^{k+1} = \beta_1 + \alpha_{11}x_1^k + \alpha_{12}x_2^k + \dots + \alpha_{1n}x_n^k \\ x_2^{k+1} = \beta_2 + \alpha_{21}x_1^{k+1} + \alpha_{22}x_2^k + \dots + \alpha_{2n}x_n^k \\ x_3^{k+1} = \beta_3 + \alpha_{31}x_1^{k+1} + \alpha_{32}x_2^{k+1} + \alpha_{33}x_3^k + \dots + \alpha_{3n}x_n^k \\ \dots \\ x_n^{k+1} = \beta_n + \alpha_{n1}x_1^{k+1} + \alpha_{n2}x_2^{k+1} + \dots + \alpha_{nn-1}x_{n-1}^{k+1} + \alpha_{nn}x_n^k \end{cases}$$

Итерационная формула имеет вид:

$$x^{k+1} = (E - B)^{-1} C x^k + (E - B)^{-1} \beta.$$

3) Полученный ответ:

A

```
[[ 21 -6 -9 -4]
 [-6 20 -4 2]
 [-2 -7 -20 3]
 [ 4 9 6 24]]
```

B

```
[[ 127]
 [-144]
 [ 236]
 [ -5]]
```

Метод итераций:

```
[[ 1.00270285]
 [-8.99953528]
 [-7.99796217]
 [ 4.99983671]]
```

Число итераций 10

Проверка $AX = B$:

```
[[ 127.03628428]
 [-144.01540064]
 [ 235.95009473]
 [ -4.97669798]]
```

Метод Зейделя:

```
[[ 0.99854895]
```

```
[-9.00123833]
[-7.99943581]
[ 5.00056517]]
Число итераций 5
```

Проверка $AX = B$:

```
[[ 126.96961946]
[-144.01718663]
[ 236.00198206]
[ -5.          ]]
```

4) Код программы:

```
def norm_matrix(matrix):
    norm = 0
    matrix_tmp = matrix.reshape(1, -1)[0]
    for i in np.arange(0, matrix_tmp.size):
        norm += matrix_tmp[i]*matrix_tmp[i]
    return np.sqrt(norm)

def get_BC(alpha):
    alpha_size = alpha[:,0].size
    B = np.zeros(alpha_size*alpha_size).reshape(alpha_size, alpha_size)
    C = np.zeros(alpha_size*alpha_size).reshape(alpha_size, alpha_size)
    for i in np.arange(0, alpha_size):
        for j in np.arange(0, alpha_size):
            if i <= j:
                C[i,j] = alpha[i,j]
                B[i,j] = 0
            else:
                C[i,j] = 0
                B[i,j] = alpha[i,j]
    return B, C

def Zedel(alpha, betta, epsilon):
    #Метод Зейделя
    k = 0
    if norm_matrix(alpha) < 1:
        x = betta.copy()
        alpha_size = alpha[:,0].size
        while True:
            pre_x = x.copy()
            for i in np.arange(0, alpha_size):
                tmp = 0
                for j in np.arange(0, alpha_size):
                    tmp += alpha[i,j]*x[j,0]
                x[i,0] = betta[i,0] + tmp
            epsilon_i = norm_matrix(pre_x - x)
```

```

        k += 1
        if epsilon_i < epsilon or k > 100:
            break

    else:
        print("Метод Зейделя не сходится")
    return x, k

def iteration(alpha, betta, epsilon):
    k = 0
    if norm_matrix(alpha) < 1:
        x = betta
        while True:
            pre_x = x
            x = betta + np.dot(alpha, pre_x)
            epsilon_i = norm_matrix(pre_x - x)
            k += 1
            if epsilon_i < epsilon or k > 100:
                break
        else:
            print("Метод итераций не сходится")
        return x, k

def get_alpha_betta(mat, answ):
    i = 0
    mat_size = mat[:,0].size
    alpha = np.zeros(mat_size*mat_size).reshape(mat_size,mat_size)
    betta = np.zeros(mat_size).reshape(-1,1)
    for row in mat:
        betta[i] = answ[i]/mat[i,i]
        alpha[i] = -row/mat[i,i]
        alpha[i,i] = 0
        i += 1
    return alpha, betta

def get_mat(file):
    mat = []
    answ = []
    with open(file) as file_handler:
        for line in file_handler:
            row = list(map(int, line.split(',')[::-1]))
            mat.append(row[::-1])
            answ.append([row[-1]])
        mat = np.asarray(mat)
        answ = np.asarray(answ).reshape(-1,1)
    return mat, answ

mat, answ = get_mat('test2.txt')
alpha, betta = get_alpha_betta(mat,answ)
epsilon = 0.01

```

```
print("A\n", mat)
print("B\n", answ)
print("Метод итераций:\n", iteration(alpha, betta, epsilon)[0], "\nЧисло итераций", iteration(alpha, betta, epsilon)[1])
print("\nПроверка AX = B:\n", np.dot(mat, iteration(alpha, betta, epsilon)[0]))
print("Метод Зейделя:\n", Zedel(alpha, betta, epsilon)[0], "\nЧисло итераций", Zedel(alpha, betta, epsilon)[1])
print("\nПроверка AX = B:\n", np.dot(mat, Zedel(alpha, betta, epsilon)[0]))
```

5) Выводы:

При большом числе уравнений прямые методы решения СЛАУ (за исключением метода прогонки) становятся труднореализуемыми на ЭВМ прежде всего из-за сложности хранения и обработки матриц большой размерности. В то же время характерной особенностью ряда часто встречающихся в прикладных задачах СЛАУ является разреженность матриц. Число ненулевых элементов таких матриц мало по сравнению с их размерностью. Для решения СЛАУ с разреженными матрицами предпочтительнее использовать итерационные методы. Метод простых итераций довольно медленно сходится. Для его ускорения существует метод Зейделя

Задание 4

1) Постановка задачи:

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.

Вариант 16:

$$A = \begin{pmatrix} 8 & -3 & 9 \\ -3 & 8 & -2 \\ 9 & -2 & -8 \end{pmatrix}$$

2) Теория:

Метод вращений Якоби применим только для симметрических матриц и решает полную проблему собственных значений и векторов. Он основан на нахождении матрицы U , которая позволяет осуществить преобразование подобия для исходной матрицы и на выходе получить диагональную матрицу с собственными значениями на главной диагонали. Алгоритм метода следующий:

- Выбираем максимальный по модулю недиагональный элемент
- Строим матрицу U таким образом, чтобы в результате преобразования подобия выбранный элемент обнулится. Для этого берем матрицу вращения, имеющую следующий вид:

$$U^k = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \end{pmatrix},$$

$$U^k = \begin{pmatrix} \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \cos \varphi^{(k)} & \dots & -\sin \varphi^{(k)} & \dots & \dots \\ \dots & & 1 & & & \\ \dots & & & \ddots & & \\ \dots & & & & 1 & \\ \dots & \sin \varphi^{(k)} & \dots & \cos \varphi^{(k)} & \dots & \dots \\ \dots & & & & & \ddots \\ 0 & & & & & & 1 \end{pmatrix}$$

Угол вращения определяется из условия равенства нулю выбранного элемента.

- выполняется преобразование подобия

В качестве критерия окончания итерационного процесса берется условие малости суммы квадратов внедиагональных элементов. Координатными столбцами собственных векторов будут столбцы матрицы $U = U^{(0)} U^{(1)} \dots U^{(k)}$

3) Полученный ответ:

```

A
[[ 8 -3  9]
 [-3  8 -2]
 [ 9 -2 -8]]
Собственные вектора
[[ 0.78338032  0.47116892 -0.40535802]
 [-0.50354661  0.86343043  0.03047448]
 [ 0.36435708  0.18024355  0.91364992]]
Собственные значения:
[ 14.11433982  5.94540849 -12.05974831]
Проверка  $A \cdot u - \lambda u = 0$ 
lambda1: [ 4.16615809e-08 -3.13208393e-09 -9.39024263e-08]
lambda2: [-3.97268246e-05  2.98663014e-06  8.95416123e-05]
lambda3: [4.60960619e-05  8.46716398e-05  1.76271931e-05]

```

4) Код программы:

```

import numpy as np
import math

def get_mat(file):
    mat = []
    with open(file) as file_handler:
        for line in file_handler:
            row = list(map(int, line.split(',')[::-1]))
            mat.append(row)
    mat = np.asarray(mat)
    return mat

def get_angle(mat, i,j):
    if mat[i,i] == mat[j,j]:
        return math.PI/4
    else:
        return 0.5 * math.atan(2*mat[i,j]/(mat[i,i] -mat[j,j]))

def get_max(mat):
    index = [0,1]
    mat_size = mat[:,0].size
    mat_tmp = np.abs(mat.copy())
    max_elem = mat_tmp[index[0],index[1]]
    for i in np.arange(0,mat_size):
        for j in np.arange(0,mat_size):
            if i==j:
                continue
            else:
                if max_elem < mat_tmp[i,j]:

```



```

        max_elem = mat_tmp[i,j]
        index = [i,j]
    else:
        pass

    return index

def norm_matrix_non_diag(matrix):
    norm = 0
    mat_size = matrix[:,0].size
    for i in np.arange(0,mat_size):
        for j in np.arange(0,mat_size):
            if i==j:
                continue
            else:
                norm += matrix[i,j]*matrix[i,j]
    return np.sqrt(norm)

def transpose(matr):
    res=[]
    n=len(matr)
    m=len(matr[0])
    for j in range(m):
        tmp=[]
        for i in range(n):
            tmp=tmp+[matr[i][j]]
        res=res+[tmp]
    return np.array(res)

epsilon = 0.001
mat = get_mat("test.txt")
mat_test = mat.copy()
mat_size = mat[:,0].size
X = np.eye(mat_size)
k = 0
while True:
    max_index = get_max(mat)
    angle = get_angle(mat, max_index[0], max_index[1])
    U = np.eye(mat_size).copy()
    U[max_index[0],max_index[0]] = math.cos(angle)
    U[max_index[0],max_index[1]] = -math.sin(angle)
    U[max_index[1],max_index[0]] = math.sin(angle)
    U[max_index[1],max_index[1]] = math.cos(angle)
    X = np.dot(X.copy(),U.copy())
    mat = np.dot(transpose(U.copy()),mat.copy())
    mat = np.dot(mat.copy(), U.copy())
    p = norm_matrix_non_diag(mat)
    k += 1
    if epsilon > p or angle == 0 or k > 100:
        break

```

```

print("A\n", mat_test)
print("Собственные вектора\n",X)
print("Собственные значения: \n",np.diagonal(mat))
print("Проверка A*u-lambda*u = 0")
print("lambda1: ", np.dot(mat_test,X[:,0])-np.diagonal(mat)[0]*X[:,0])
print("lambda2: ", np.dot(mat_test,X[:,1])-np.diagonal(mat)[1]*X[:,1])
print("lambda3: ", np.dot(mat_test,X[:,2])-np.diagonal(mat)[2]*X[:,2])

```

5) Выводы:

Метод вращений Якоби применим только для симметрических матриц и решает полную проблему собственных значений и собственных векторов таких матриц

Задание 5

1) Постановка задачи:

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.

Вариант 16:

$$\begin{pmatrix} 1 & 2 & 5 \\ -8 & 0 & -6 \\ 7 & -9 & -7 \end{pmatrix}$$

2) Теория:

QR-алгоритм позволяет находить как вещественные, так и комплексные собственные значения. В основе алгоритма лежит разложение исходной матрицы на произведение ортогональной матрицы Q и верхней

треугольной матрицы R. Алгоритм использует следующий итерационный процесс:

$$A^{(0)} = A,$$

$$A^{(0)} = Q^{(0)} R^{(0)} \text{ - производится QR - разложение,}$$

$$A^{(1)} = R^{(0)} Q^{(0)} \text{ - производится перемножение матриц,}$$

.....

$$A^{(k)} = Q^{(k)} R^{(k)} \text{ - разложение,}$$

$$A^{(k+1)} = R^{(k)} Q^{(k)} \text{ перемножение.}$$

Если в ходе итерационного процесса прослеживается комплексно-сопряженная пара собственных значений, то начиная с некоторой итерации они будут отличаться незначительно. Поэтому в качестве

окончания итерационного процесса можно взять условие $|\lambda^{(k)} - \lambda^{(k-1)}| \leq \varepsilon$.

Само QR разложение выполняется при помощи матрицы Хаусхолдера,

имеющей следующий вид: $H = E - \frac{2}{v^T v} v v^T$, где вектор v определяется таким образом, чтобы в результате преобразования $A_i = H_i A_{i-1}$ обнулить поддиагональные элементы. Повторив данное преобразование n-1 раз, получим QR разложение, где $Q = (H_{n-1} H_{n-2} \dots H_0)^T = H_1 H_2 \dots H_{n-1}$, $R = A_{n-1}$.

3) Полученный ответ:

```
A =
[[ 1  2  5]
 [-8  0 -6]
 [ 7 -9 -7]]
A1 =
[[ 5.52631579 -6.84843428  8.3270333 ]
 [-1.38630894 -11.82117255 -4.30467015]
 [ 0.93344237  1.03386898  0.29485676]]
A2 =
[[ 7.84210526 -2.99375662 -7.57675573]
 [ 2.85617311 -12.43207228  4.04282734]
 [ 0.23278001 -0.17492961 -1.41003298]]
A3 =
[[ 5.26274066e+00 -9.48769139e+00  5.94161177e+00]
 [-3.46731410e+00 -1.00269593e+01 -6.47255690e+00]]
```

```

[ 3.44673542e-02 -2.57219288e-03 -1.23578138e+00]]
A4 =
[[ 6.63310460e+00  1.45145956e+00 -8.48065925e+00]
 [ 7.46658569e+00 -1.13490885e+01  2.18475355e+00]
 [ 7.02232632e-03 -1.00010379e-03 -1.28401612e+00]]
A5 =
[[ 1.00787820e+00 -1.24593921e+01  4.00689517e+00]
 [-6.44077652e+00 -5.73054920e+00 -7.79346869e+00]
 [ 8.98114653e-04 -6.35181255e-04 -1.27732900e+00]]
A6 =
[[ -2.68146003e+00  7.01900306e+00 -8.31882382e+00]
 [ 1.30372730e+01 -2.03982494e+00 -2.75209972e+00]
 [ 1.76162737e-04  1.11420420e-04 -1.27871503e+00]]
A7 =
[[ -6.02356238e+00 -1.23498593e+01 -1.02000382e+00]
 [-6.33149891e+00  1.30205310e+00  8.70284457e+00]
 [-1.69210737e-05  3.09685699e-05 -1.27849073e+00]]
A8 =
[[ -1.15073555e+01  1.11442921e+00 -5.60221788e+00]
 [ 7.13277672e+00  6.78588834e+00 -6.73754706e+00]
 [ 2.47556899e-06  5.92503197e-06 -1.27853280e+00]]
A9 =
[[ -1.01228681e+01 -9.36638002e+00  1.21202457e+00]
 [-3.34802988e+00  5.40139372e+00  8.67816090e+00]
 [-2.33780497e-07  1.09537251e-06 -1.27852558e+00]]
A10 =
[[ -1.23826531e+01 -3.48449696e+00 -3.87575273e+00]
 [ 2.53385274e+00  7.66117999e+00 -7.85862611e+00]
 [ 2.80332044e-08  1.98554930e-07 -1.27852689e+00]]
A11 =
[[ -1.13903754e+01 -7.38300210e+00  2.22161201e+00]
 [-1.36465231e+00  6.66890208e+00  8.47607849e+00]
 [-2.83570735e-09  3.52311939e-08 -1.27852666e+00]]
A12 =
[[ -1.21680278e+01 -5.12619676e+00 -3.21412443e+00]
 [ 8.92153013e-01  7.44655447e+00 -8.15161764e+00]
 [ 3.16037367e-10  6.28150043e-09 -1.27852670e+00]]
A13 =
[[ -1.17543708e+01 -6.53399790e+00  2.60944790e+00]
 [-5.15648128e-01  7.03289754e+00  8.36482203e+00]
 [-3.31179802e-11  1.11230394e-09 -1.27852669e+00]]
A14 =
[[ -1.20269493e+01 -5.69786856e+00 -2.97354133e+00]
 [ 3.20481212e-01  7.30547600e+00 -8.24242185e+00]
 [ 3.59879216e-12  1.97537438e-10 -1.27852670e+00]]
A15 =
[[ -1.18700427e+01 -6.20883717e+00  2.75292893e+00]
 [-1.90487391e-01  7.14856942e+00  8.31870472e+00]
 [-3.82434403e-13  3.50021586e-11 -1.27852670e+00]]
A16 =
[[ -1.19678144e+01 -5.90206236e+00 -2.88605377e+00]

```

```

[ 1.16287417e-01  7.24634109e+00 -8.27346116e+00]
[ 4.11868480e-14  6.20952910e-12 -1.27852670e+00]]
A17 =
[[-1.19097873e+01 -6.08819632e+00  2.80553076e+00]
 [-6.98465431e-02  7.18831397e+00  8.30111216e+00]
 [-4.39980077e-15  1.10073082e-12 -1.27852670e+00]]
A18 =
[[-1.19452439e+01 -5.97598500e+00 -2.85416466e+00]
 [ 4.23647703e-02  7.22377056e+00 -8.28451628e+00]
 [ 4.72314566e-16  1.95209007e-13 -1.27852670e+00]]
A19 =
[[-1.19239590e+01 -6.04389397e+00  2.82476519e+00]
 [-2.55441960e-02  7.20248565e+00  8.29458664e+00]
 [-5.05525874e-17  3.46095746e-14 -1.27852670e+00]]
A20 =
[[-1.19368734e+01 -6.00289253e+00 -2.84252781e+00]
 [ 1.54572483e-02  7.21540012e+00 -8.28851625e+00]
 [ 5.42040479e-18  6.13712883e-15 -1.27852670e+00]]

lambda0: -11.945243860830686

lambda1: 7.2024856496655385

lambda2: -1.2785266959322459

```

4) Код программы:

```

import numpy as np

def get_mat(file):
    mat = []
    answ = []
    with open(file) as file_handler:
        for line in file_handler:
            row = list(map(int, line.split(',')[ :-1]))
            mat.append(row)
    return np.array(mat)

def get_v(column, size, k):
    v = np.zeros(size)
    v[k] = column[k]+np.sign(column[k])*np.sqrt(np.sum(column[k:]*column[k:]))
    for i in range(k+1, size):
        v[i] = column[i]
    return v

def get_H(column, size, k):

```

```

v = get_v(column,size,k)[: , np.newaxis]
return np.eye(size) - 2*np.dot(v,v.T)/np.dot(v.T,v)

def get_A(H,A0):
    return np.dot(H,A0)

def get_QR(A):
    size = A[:,1].size
    Q = np.eye(size)
    for i in range(size-1):
        H = get_H(A[:,i],A[:,i].size,i)
        A = get_A(H,A)
        Q = np.dot(Q,H)
    return Q,A

def solve_equation(A, i):
    size = A[:,1].size
    if i+1 < size:
        a12 = A[i][i + 1]
        a21 = A[i + 1][i]
        a22 = A[i + 1][i + 1]
    else:
        a12 = 0
        a21 = 0
        a22 = 0
    a11 = A[i][i]
    a = 1
    b = -a11 - a22
    c = a11 * a22 - a12 * a21
    D = b*b-4*a*c
    if D < 0:
        l1 = (-b+np.sqrt(complex(D)))/2
        l2 = (-b-np.sqrt(complex(D)))/2
    else:
        l1 = (-b+np.sqrt(D))/2
        l2 = (-b-np.sqrt(D))/2
    return np.array([l1,l2])

def have_complex_lambda(A, eps, i):
    Q, R = get_QR(A)
    A1 = np.dot(R,Q)
    lambda1 = solve_equation(A, i)
    lambda2 = solve_equation(A1, i)
    if np.all(abs(lambda1 - lambda2) <= eps) and isinstance(lambda1[0],
complex) and isinstance(lambda2[0], complex):
        return True
    else:
        return False

```

```

def get_eig(A,eps,i,k):
    while True:
        Q, R = get_QR(A)
        A = np.dot(R,Q)
        k+=1
        print("A",k, " =", sep="")
        print(A)
        if np.sqrt(np.sum(A[i + 1:, i]*A[i + 1:, i])) <= eps:
            return A[i,i], False, A, k
        elif np.sqrt(np.sum(A[i + 2:, i]*A[i + 2:, i])) <= eps and
have_complex_lambda(A, eps, i):
            return solve_equation(A,i), True, A, k

def QR(A, eps):
    eigs = []
    i = 0
    k = 0
    c = 0
    size = A[:,1].size
    while i < size:
        eig = get_eig(A,eps,i,k)
        k = eig[3]
        if eig[1]:
            eigs.append(eig[0])
            i+=2
        else:
            eigs.append(eig[0])
            i+=1
        A = eig[2]
    for l in eigs:
        if l.size == 1:
            print("\nlambda",c,": ", l, sep="")
            c+=1
        else:
            print("\nlambda",c,": ", l[0], sep="")
            c+=1
            print("\nlambda",c,": ", l[1], sep="")
            c+=1

A = get_mat("test2.txt")
print("A = \n", A)
QR(A,0.05)

```

5) Выводы:

QR-алгоритм позволяет находить как вещественные, так и комплексные собственные значения. В основе алгоритма лежит разложение исходной матрицы на произведение ортогональной матрицы Q и верхней треугольной матрицы R.