Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ "МЭИ"

Институт информационных и вычислительных технологий

Кафедра математического и компьютерного моделирования

Курсовая работа

по дисциплине "Методы вычислительной математики"

Студент: Симаков А.М.
Преподаватель: Вестфальский А.Е.

Москва 2023

1 Постановка задачи

Как Вы знаете, одной из важных задач в линейной алгебре стоит проблема вычисления собственных значений матриц.

В настоящее время лучшим методом вычисления всех собственных значений квадратных заполненных матриц общего вида (умеренного порядка) является QR-алгоритм.

Процедура построения, вопросы ускорения алгоритма были доложены на лекции

2 Тесты

1.

$$A = \begin{pmatrix} -1 & -6 \\ 2 & 7 \end{pmatrix}$$

Собственные числа: $\lambda_1 = 1, \lambda_2 = 5$
Собственные числа, полученные QR-алгоритмом: $\lambda_1 = 1, \lambda_2 = 5$

2.

$$A = \begin{pmatrix} 5 & 6 & 3 \\ -1 & 0 & 1 \\ 1 & 2 & -1 \end{pmatrix}$$

Собственные числа: $\lambda_1 = -2, \lambda_2 = 4, \lambda_3 = 2$
Собственные числа, полученные QR-алгоритмом: $\lambda_1 = -2, \lambda_2 = 4, \lambda_3 = 2$

3.

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

Собственные числа: $\lambda_1 = 0.5 + 1.323j, \lambda_2 = 0.5 - 1.323j$
Собственные числа, полученные QR-алгоритмом: $\lambda_1 = 0.5 + 1.323j, \lambda_2 = 0.5 - 1.323j$

4.

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \end{pmatrix}$$

Собственные числа: $\lambda_1 = 17.165, \lambda_2 = -1.165, \lambda_3 = \lambda_4 = 0$
Собственные числа, полученные QR-алгоритмом: $\lambda_1 = 17.165, \lambda_2 = -1.165, \lambda_3 = \lambda_4 = 0$

3 Программный код

```
import numpy as np
import time
import random
import scipy
np.set_printoptions(suppress=True)
np.set_printoptions(precision=6, floatmode='fixed')
from math import hypot
import random

def createMtr(num, n=0):

    if num == 1:
      A = np.zeros((n, n))
      for i in range(len(A)):
          for j in range(len(A[i])):
              A[i, j] = i + j + 1

    elif num == 2:
        A = np.array([[5, 6, 3],
                      [-1, 0, 1],
                      [1, 2, -1]])

    elif num == 3:
        A = np.array([[-1, -6],
                      [2, 7]])

    elif num == 4:
        A = np.array([[-3, -4],
                      [2, 2]])


    return A


A = createMtr(1, 4)
print('Матрица A')
print(A)

#сумма квадратов элементов ниже главной диагонали
def sumSqBelow(A):
    sum = 0
    n, m = np.shape(A)
    rows, cols = np.tril_indices(n, -1, m)
    for (row, col) in zip(rows, cols):
         sum += (A[row, col])**2

    return sum
```

```
def norm(a, b):
    sum = 0
    a = np.sort(a)
    b = np.sort(b)
    for i in range(len(a)):
        sum += (a[i]-b[i])**2
    return sum

#print(sumSqBelow(A))

def houseVector(a):
    v = a / (a[0] + np.copysign(np.linalg.norm(a), a[0]))
    v[0] = 1
    tau = 2 / (v.T @ v)

    return v,tau


def Householder(A: np.ndarray):
    m, n = A.shape
    R = A.copy()
    Q = np.identity(m)

    for j in range(0, n):
        # Apply Householder transformation.
        v, tau = houseVector(R[j:, j, np.newaxis])

        H = np.identity(m)
        H[j:, j:] -= tau * (v @ v.T)
        R = H @ R
        Q = H @ Q

    return Q[:n].T, np.triu(R[:n])



#QR-алгоритм
#def QRalg(A):
  #  A1 = np.copy(A)
 #   start_time = time.time()
  #  while True:
   #     Q, A1 = np.linalg.qr(A1)
     #   A1 = A1 @ Q
    #    if sumSqBelow(A1) < 10**(-12):
     #       break
   #end_time = time.time()
   # return A1, end_time - start_time
```

```
def QRalg(A):
tol = 10**(-12); cnttol = 500
m = A.shape[0]; eig = np.zeros(m, dtype=complex)
n = m - 1

while n > 0:
cnt = 0

while max(abs(A[n, 0:n])) > tol and cnt < cnttol:
cnt += 1
Q, R = np.linalg.qr(A)
A = R @ Q

if cnt < cnttol:
eig[n]=A[n, n]
n -= 1
A = A[:n+1, :n+1]

else:
disc = complex(( A[-2, -2] - A[-1, -1] )**2 + 4*A[-1, -2]*A[-2, -1])
eig[n]=( A[-2, -2] + A[-1, -1] + np.sqrt(disc) )/2.
eig[n-1]=( A[-2, -2] + A[-1, -1] - np.sqrt(disc) )/2.
n -= 2
A = A[:n+1, :n+1]

if n==0:
eig[0] = A[0,0]

return eig

print('Собственные числа матрицы A с помощью втроенной функции: ')
a, b = np.linalg.eig(A)
print(a, '\n')
s = time.time()
eig = QRalg(A)
en = time.time()
#print('Матрица после QR-алгоритма:')
#print(A1)
print('Собственные числа через QR-алгоритм: ')
print(eig, '\n')

print('Норма разности:', norm(a, eig), '\n')

print('Время, затраченное на QR-алгоритм:', en-s)

#сигнум
def sign(x):
    if x > 0:
      res = 1
```

```
    elif x < 0:
      res = -1
    else:
      res = 0
    return res

#функция создания орта
def ort(size, pos):
    if size == 1:
       return 1
    else:
       e = np.zeros(size)
       e[pos] = 1
       return e

#приведение к форме Хессенберга
def hessenberg(A1):
    A = np.copy(A1)

    for i in range(len(A)-3):
        x = A[i+1:, i]
        #print(x)
        u = np.zeros(len(x))
        u = x + sign(x[0]) * np.linalg.norm(x, 2) * ort(len(x), 0)
        #print(u)
        P = np.eye(len(x)) - 2 * ((np.outer(u, u.transpose()))/(np.dot(u.transpose(), u)))
        #print((np.outer(u, u.transpose())))
        #print(np.dot(u.transpose(), u))
        #print(P)
        P1 = np.zeros((len(A), len(A)))
        P1[i+1:, i+1:] = P
        for i in range(len(P)):
            if P1[i, i] == 0:
               P1[i, i] = 1
            else:
              break
        #print(P1)

        A = P1.T @ A @ P1
        #print(A)
        #print('\n')

    return A


#print('Матрица A')
#print(A, '\n')

H1 = hessenberg(np.copy(A))
print('Матрица в форме Хессенберга')
```

```
#print(H1, '\n')

print('Матрица в форме Хессенберга (встроенная функция)')
#print(scipy.linalg.hessenberg(np.copy(A)))


print('Собственные числа матрицы A с помощью встроенной функции: ')
print(a, '\n')

s = time.time()
eig = QRalg(H1)
en = time.time()
#print('Матрица H после QR-алгоритма')
#print(H1)
print('Собственные числа через QR-алгоритм: ')
print(eig, '\n')
print('Норма разности:', norm(a, eig), '\n')
print('Время, затраченное на QR-алгоритм:', en-s)

H = hessenberg(A)

def shiftedqr(A):
tol = 10**(-12); cnttol = 500
m = A.shape[0]; eig = np.zeros(m, dtype=complex)
n = m - 1

while n > 0:
cnt = 0

while max(abs(A[n, 0:n])) > tol and cnt < cnttol:
cnt += 1      # keep track of number of qr's
mu = A[n, n] # define shift mu
Q, R = np.linalg.qr( A - mu*np.eye(n+1) )
A = R @ Q + mu*np.eye(n + 1)

if cnt < cnttol: # have isolated 1x1 block
eig[n]=A[n, n] # declare eigenvalue
n -= 1 # decrement n
A = A[:n+1, :n+1] # deflate by 1

else: # have isolated 2x2 block
disc = complex(( A[-2, -2] - A[-1, -1] )**2 + 4*A[-1, -2]*A[-2, -1])
eig[n]=( A[-2, -2] + A[-1, -1] + np.sqrt(disc) )/2.
eig[n-1]=( A[-2, -2] + A[-1, -1] - np.sqrt(disc) )/2.
n -= 2
A = A[:n+1, :n+1] # deflate by 2

if n==0:
eig[0] = A[0,0] # only a 1x1 block remains
```

```
return eig

start_time = time.time()
eig = shiftedqr(H)
end_time = time.time()

print('Собственные числа матрицы A с помощью втроенной функции: ')
print(a, '\n')
print('Собственные числа через QR-алгоритм со сдвигами: ')
print(eig, '\n')
print('Норма разности:', norm(a, eig), '\n')
print('Время, затраченное на QR-алгоритм со сдвигами:', end_time-start_time)
```