**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 линейной алгебры».

Студент: Шевчук П.В.

Группа: 80-304Б

Вариант: 13

Оценка:

Москва, 2019

Постановка задачи.

Реализовать методы для

решения СЛАУ:

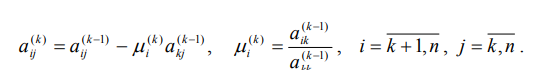
1. LU-разложение
2. Метод прогонки
3. Метод простых итераций (3.1) и метод Зейделя (3.2)

нахождения собственных векторов и собственных значений:

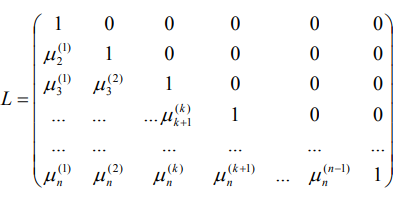
1. Метод вращений (Якоби)
2. QR-алгоритм

Описание методов.

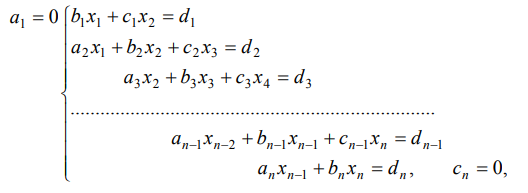
1. Матрица А разлагается на произведение LU, где L – нижняя треугольная, U – верхняя треугольная. Для вычисления U выполнятся прямой ход метода Гаусса для которого вычисляются коэффициенты μ.



Из этих коэффициентов строится матрица L.



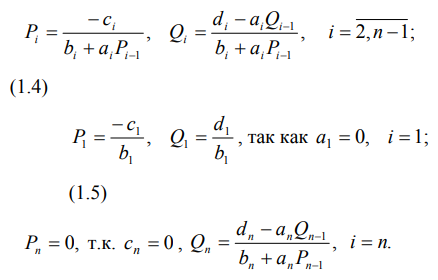
Матрица будет искомой U. Далее решается система Lz = b, а затем система Ux = z. Эти действия эквивалентны обратному ходу метода Гаусса. Время работы O(n^3).

1. А – трехдиагональная матрица. b – коэффициенты элементов главной диагонали, a – под главной, c – над, d – элементы вектора. 

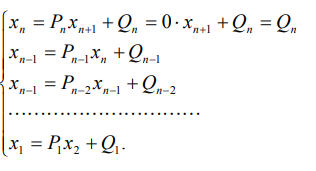
Решение будем искать в виде

.

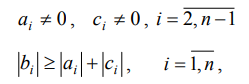


В ходе прямого хода определяются прогоночные коэффициенты P и Q. 

Затем обратным ходом вычисляется искомый вектор x.

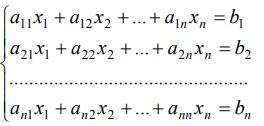


Для устойчивости необходимо:

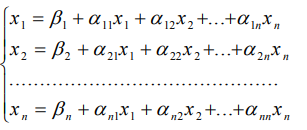


И строгое неравенство достигается хотя бы при одном i. Время работы O(n).

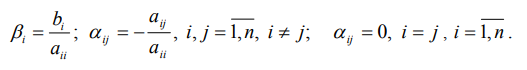
1. (3.1) Применяется преимущественно для разреженных матриц. СЛАУ



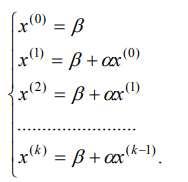
приводится к эквивалентному виду



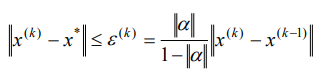
Один из способов такого приведения – способ Якоби. Поменяем строки в матрице так, чтобы на диагонали не было нулей. Далее



Используя его, получаем итерационную формулу для метода простых итераций



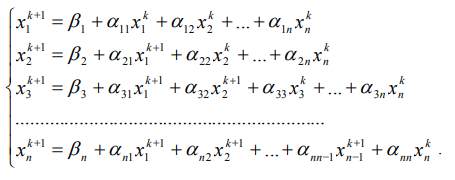
Оценка погрешности



Достаточное условие сходимости .



(3.2) Применяется преимущественно для разреженных матриц. Является ускоренной версией предыдущего алгоритма. Ускорение достигается благодаря использованию на текущей итерации элементов вектора решения, уже вычисленных на текущей итерации (значения остальных компонент берутся из предыдущей итерации).



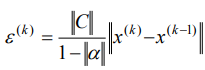
Матрица α представляется в виде B + C, где B – нижняя треугольная(b[i][i] == 0), C – верхняя треугольная.

Итерационная формула:

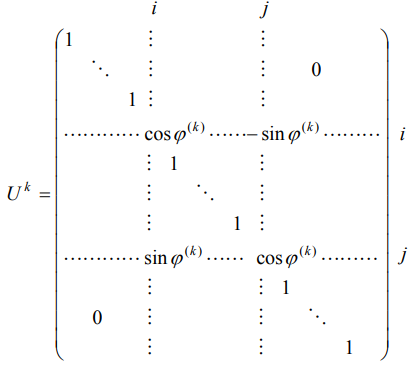
C:\Users\prost(polzovatel)\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Скриншот 04-04-2018 210656.png

Достаточное условие сходимости . Оценка погрешности

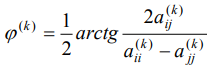
.



1. А – симметрическая. Ищется преобразование подобия C:\Users\prost(polzovatel)\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Скриншот 04-04-2018 221505.png, где Λ – диагональная. Ее диагональные элементы будут собственными значениями. Векторы матрицы U – собственными векторами. На каждой итерации ищется максимальный по модулю элемент (его мы будем обнулять), в соответствии с ним строится матрица вращения.



Угол выбирается следующим образом . Итерационная формулаC:\Users\prost(polzovatel)\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Скриншот 04-04-2018 221714.png. Критерий останова – корень суммы квадратов поддиагональных элементов меньше заданной точности. Собственные значения – это диагональные элементы конечной матрицы A. Собственные векторы – векторы матрицы C:\Users\prost(polzovatel)\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Скриншот 04-04-2018 222129.png.



1. Ищется представление А = QR, где Q –ортогональная, R – верхняя треугольная, с помощью преобразования Хаусхолдера. Затем матрицы R и Q перемножаются в обратном порядке. То есть каждая итерация состоит из двух шагов A[k] = Q[k]R[k], A[k + 1] = R[k]Q[k]. Итерации продолжаются пока: для вещественных собственных значений – сумма поддиагональных элементов столбца не станет меньше или равной заданной точности, для комплексных |λ[k] – λ[k -1]| не станет меньше заданной точности. Комплексные λ находятся из решений квадратных уравнений, вещественные стоят на диагонали последней полученной матрицы А. Критерий сходимости C:\Users\prost(polzovatel)\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Скриншот 04-04-2018 223018.png.



Общая информация.

Данная работа состоит из 5 модулей, которые позволяют решать различные задачи решения СЛАУ и нахождения собственных векторов и собственных значений. Полученные в ходе расчетов результаты сохраняются в отдельный файл. Поскольку составленные программы могут обрабатывать любой корректный ввод (в том числе все варианты заданий из лабораторных работ), они могут служить удобным примером для реализации собственных решателей, применяясь для сравнения результатов. Что касается технических деталей реализации, все программы написаны на языке python.

Запуск программы.

Чтобы воспользоваться программой, необходимо скомпилировать файл Lw и посмотреть полученный файл out:

*python .\Lw1\_1.py --input test1 --output out1*

*cat out1*

Входные данные и результаты.

Вариант 1.

|  |  |
| --- | --- |
| **Test1**  4  -6 -5 -3 -8  5 -1 -5 -4  -6 0 5 5  -7 -2 8 5  101 51 -53 -63 | **out1**  Answer:  x = [-2. -3. -6. -7.]  inverse\_matrix PA:  array([[ 0.14492754, -0.07246377, -0.20289855, 0.07246377],  [-0.2115942 , 0.0057971 , -0.26376812, -0.6057971 ],  [ 0.24057971, -0.02028986, -0.57681159, -0.37971014],  [-0.06666667, -0.06666667, 0.53333333, 0.46666667]])  det = -345.0Determinant: 8 |
| **Test2**  5  14 9  -8 14 6  -5 -17 8  1 5 -2  -4 -10  125 -56 144 36 70 | **Out2**  Answer:  x1 = 7.0  x2 = 3.0  x3 = -7.0  x4 = 5.0  x5 = -9.0 |
| **Test3**  4  24 -7 -4 4  -3 -9 -2 -2  3 7 24 9  1 -6 -2 -15  -190 -12 155 -17  0.001 | **Out3**  Iter:  Steps = 10  x = [-5.99999762 2.00004091 7.00005338 -0.99997317]  Seidel:  Steps = 6  x = [-5.99990367 1.99993908 7.00002904 -0.99997308] |
| **Test4**  3  8 0 -2  0 5 4  -2 4 -6  0.001 | **Out4**  Solution:  Steps = 5  Eigenvalues:  [ 8.411 6.123 -7.534]  Eigenvectors:  [[ 0.953 0.277 0.122]  [-0.23 0.925 -0.302]  [-0.196 0.26 0.946]] |
| **Test5**  3  -1 2 9  9 3 4  8 -4 -6  0.001 | **Out5**  Solution:  Steps = 11  Eigenvalues:  [-13.006+0.j 4.503+2.804j 4.503-2.804j] |

Выводы:

В ходе выполнения работы были получены оценки сложности и скорости сходимости методов решения систем линейных уравнений:

1)Метод Гаусса требует O(n^3) операций.

2)Метод прогонки – O(n).

3)Метод простых итераций – скорость сходимости равна скорости сходимости геометрической прогрессии со знаменателем .

4)Метод Зейделя – скорость сходимости равна скорости сходимости геометрической прогрессии со знаменателем .

Полученные оценки наводят нас на следующие выводы: для систем с трехдиагональной матрицей коэффициентов, которые удовлетворяют условиям применения метода прогонки, лучше использовать именно его; для разреженных матриц лучше применять метод Зейделя, ведь он почти всегда быстрее метода простых итераций; во всех остальных случаях поможет метод Гаусса.

Также при нахождении собственных значений для симметрических матриц выгоднее применять метод вращений Якоби, для других – QR алгоритм.

Исходный код.

import argparse # python .\Lw1\_1.py --input test1 --output out1

import numpy as np

from pprint import pformat

class Matrix:

def \_\_init\_\_(self, input\_name, output\_name, log\_name):

self.log\_file = log\_name

self.answer\_file = output\_name

self.shape, self.matrix, self.b = self.get\_matrix(input\_name)

self.l\_matrix, self.u\_matrix, self.p = self.lu\_decomposition()

self.x = self.solve\_by\_lu(self.create\_p\_matrix(self.b, self.p))

self.inverse\_matrix = self.inverse()

self.det\_matrix = self.det()

@staticmethod

def get\_matrix(filename):

with open(filename) as f:

shape = int(f.readline())

matrix = np.array([[int(num) for num in line.split()]

for \_, line in zip(range(shape), f)])

b = np.array([int(num) for num in f.readline().split()])

return shape, matrix, b

@staticmethod

def swap\_lines(matrix, l1, l2):

matrix[l1], matrix[l2] = matrix[l2].copy(), matrix[l1].copy()

def modificate(self):

p = []

mtrx = self.matrix.copy()

for j in range(self.shape - 1):

tmp = [(mtrx[i][j], i) for i in range(j, self.shape)]

idx = max(tmp, key=lambda x: abs(x[0]))[1]

if idx != j:

p.append((j, idx))

self.swap\_lines(mtrx, j, idx)

return p, mtrx

def create\_p\_matrix(self, matrix, p):

p\_matrix = matrix.copy()

for i, j in p:

self.swap\_lines(p\_matrix, i, j)

return p\_matrix

def lu\_decomposition(self):

l\_matrix = np.zeros((self.shape, self.shape))

u\_matrix = np.zeros((self.shape, self.shape))

p, p\_matrix = self.modificate()

for j in range(self.shape):

l\_matrix[j][j] = 1.0

for i in range(j + 1):

s1 = sum(u\_matrix[k][j] \* l\_matrix[i][k] for k in range(i))

u\_matrix[i][j] = p\_matrix[i][j] - s1

for i in range(j, self.shape):

s2 = sum(u\_matrix[k][j] \* l\_matrix[i][k] for k in range(j))

l\_matrix[i][j] = (p\_matrix[i][j] - s2) / u\_matrix[j][j]

return l\_matrix, u\_matrix, p

def solve\_by\_lu(self, b):

z = np.zeros(self.shape)

x = np.zeros(self.shape)

z[0] = b[0]

for i in range(1, self.shape):

s1 = sum([self.l\_matrix[i][j] \* z[j] for j in range(i)])

z[i] = b[i] - s1

x[-1] = z[-1] / self.u\_matrix[-1][-1]

for i in reversed(range(self.shape - 1)):

s2 = sum([self.u\_matrix[i][j] \* x[j]

for j in range(i + 1, self.shape)])

x[i] = (z[i] - s2) / self.u\_matrix[i][i]

return x

def inverse(self):

return np.transpose(np.array([self.solve\_by\_lu(col)

for col in np.eye(self.shape)]))

def det(self):

return (-1)\*\*len(self.p) \* np.prod(np.diag(self.u\_matrix))

def print\_answer(self):

if self.log\_file:

with open(self.log\_file, 'w') as fl:

fl.write(f'Shape = {self.shape}\nMatrix:\n')

fl.write(pformat(self.matrix.round(2)))

fl.write(f'\nVector = {self.b}\nL-matrix:\n')

fl.write(pformat(self.l\_matrix.round(2)))

fl.write('\nU-matrix:\n')

fl.write(pformat(self.u\_matrix.round(2)) + '\n')

with open(self.answer\_file, 'w') as fo:

fo.write('Answer:\n')

fo.write(f'x = {self.x.round(2)}\n')

fo.write(f'inverse\_matrix PA:\n{pformat(self.inverse\_matrix)}\n')

fo.write(f'det = {round(self.det\_matrix)}\n')

fo.write(f'\nCheck by NumPy:\nx = ')

fo.write(f'{np.linalg.solve(self.matrix, self.b).round(2)}\n')

fo.write(f'inverse\_matrix A:\n')

fo.write(f'{pformat(np.linalg.inv(self.matrix))}\n')

fo.write(f'det = {round(np.linalg.det(self.matrix))}\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

parser.add\_argument('--log', help='Logging')

args = parser.parse\_args()

matrix = Matrix(args.input, args.output, args.log)

matrix.print\_answer()

if \_\_name\_\_ == "\_\_main\_\_":

main()

import argparse # python .\Lw1\_2.py --input test1 --output out1

from copy import deepcopy

from numpy.linalg import solve

from pprint import pformat

def get\_matrix(filename):

with open(filename) as f:

shape = int(f.readline())

matrix = [[int(num) for num in line.split()]

for \_, line in zip(range(shape), f)]

matrix[0].insert(0, 0)

matrix[-1].append(0)

d = [int(num) for num in f.readline().split()]

return shape, matrix, d

def tma(matrix, d, shape, fl):

a, b, c = zip(\*matrix)

p = [-c[0] / b[0]]

q = [d[0] / b[0]]

x = [0] \* (shape + 1)

for i in range(1, shape):

p.append(-c[i] / (b[i] + a[i] \* p[i - 1]))

q.append((d[i] - a[i] \* q[i - 1]) / (b[i] + a[i] \* p[i - 1]))

for i in reversed(range(shape)):

x[i] = p[i] \* x[i + 1] + q[i]

if fl:

fl.write(f'a = {a}\nb = {b}\nc = {c}\nP = {p}\nQ = {q}\n')

fl.write(f'x = {x[:-1]} (from n to 1)\n')

return x[:-1]

def full\_matrix(matrix):

new\_matrix = deepcopy(matrix)

new\_matrix[0].pop(0)

new\_matrix[-1].pop()

return [[0] \* (idx - 1) + line + [0] \* (3 - idx)

for idx, line in enumerate(new\_matrix)]

def print\_answer(x, fo):

for idx, val in enumerate(x):

fo.write(f'x{idx + 1} = {val:.2}\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

parser.add\_argument('--log', help='Logging')

args = parser.parse\_args()

shape, matrix, d = get\_matrix(args.input)

fl = None

fo = open(args.output, 'w')

if args.log:

fl = open(args.log, 'w')

fl.write(f'Shape = {shape}\nMatrix:\n')

fl.write(pformat(full\_matrix(matrix)))

fl.write(f'\nVector d = {d}\n')

x = tma(matrix, d, shape, fl)

fo.write('Answer:\n')

print\_answer(x, fo)

fo.write('\nCheck by NumPy:\n')

num\_x = solve(full\_matrix(matrix), d)

print\_answer(num\_x, fo)

if fl:

fl.close()

fo.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

import argparse # python .\Lw1\_3.py --input test1 --output out1

import numpy as np

from numpy.linalg import norm, solve, inv

def get\_matrix(filename):

with open(filename) as f:

shape = int(f.readline())

matrix = np.array([[float(num) for num in line.split()]

for \_, line in zip(range(shape), f)])

b = np.array([int(num) for num in f.readline().split()])

eps = float(f.readline())

return shape, matrix, b, eps

class Matrix:

def \_\_init\_\_(self, input\_name, output\_name, log\_name):

self.out\_file = output\_name

self.log\_file = log\_name

if log\_name:

open(log\_name, 'w').close()

self.shape, self.matrix, self.b, self.eps = get\_matrix(input\_name)

self.alpha, self.beta = self.equivalent\_form()

self.k\_iter, self.x\_iter = self.iter\_solve()

self.k\_sdl, self.x\_sdl = self.seidel\_solve()

def equivalent\_form(self):

alpha = np.zeros((self.shape, self.shape))

beta = []

for i in range(self.shape):

beta.append(self.b[i] / self.matrix[i][i])

for j in range(self.shape):

if i != j:

alpha[i][j] = -self.matrix[i][j] / self.matrix[i][i]

return alpha, np.array(beta)

def iter\_solve(self):

print(norm(self.alpha, np.inf))

x = self.beta

k = 0

nec\_norm = self.get\_norm() # норма вектора в С

if self.log\_file:

f\_log = open(self.log\_file, 'a')

f\_log.write('Iter:\n')

while True:

x\_next = self.beta + self.alpha @ x

k += 1

if self.log\_file:

f\_log.write(f'eps{k} = {nec\_norm(x\_next, x):.4f}\n')

f\_log.write(f'x{k} = {x\_next.round(4)}\n')

if nec\_norm(x\_next, x) <= self.eps:

break

x = x\_next

if self.log\_file:

f\_log.close()

return k, x\_next

def seidel\_solve(self):

k = 0

b = np.tril(self.alpha, -1) # обнулить элементы выше k-ой диагонали

c = self.alpha - b

tmp1 = inv(np.eye(self.shape, self.shape) - b) @ c # альфа

tmp2 = inv(np.eye(self.shape, self.shape) - b) @ self.beta # вектор правых частей

x = tmp2

nec\_norm = self.get\_norm(tmp1, c)

if self.log\_file:

f\_log = open(self.log\_file, 'a')

f\_log.write('Seidel:\n')

while True:

x\_next = tmp2 + tmp1 @ x

k += 1

if self.log\_file:

f\_log.write(f'eps{k} = {nec\_norm(x\_next, x):.4f}\n')

f\_log.write(f'x{k} = {x\_next}\n')

if nec\_norm(x\_next, x) <= self.eps:

break

x = x\_next

if self.log\_file:

f\_log.close()

return k, x\_next

def get\_norm(self, s\_alpha=None, c=None):

for i in range(self.shape):

tmp = 0

for j in range(self.shape):

if i != j:

tmp += abs(self.matrix[i][j])

if abs(self.matrix[i][i]) <= tmp:

return lambda x, y: norm(x - y, np.inf)

if c is not None:

method1 = s\_alpha

method2 = c

else:

method1 = self.alpha

method2 = self.alpha

coef = norm(method2, np.inf) / (1 - norm(method1, np.inf)) # при выполнении достаточного условия

return lambda x, y: coef \* norm(x - y, np.inf)

def print\_solution(self):

if self.log\_file:

with open(self.log\_file, 'a') as f\_log:

f\_log.write(f'General EPS = {self.eps}\nMatrix info:\n')

f\_log.write(f'Shape = {self.shape}\n')

f\_log.write(f'Matrix:\n{self.matrix}\n')

f\_log.write(f'Alpha:\n{self.alpha}\n')

f\_log.write(f'Beta:\n{self.beta}\n')

with open(self.out\_file, 'w') as f\_out:

f\_out.write(f'Iter:\nSteps = {self.k\_iter}\nx = {self.x\_iter}\n')

f\_out.write(f'Seidel:\nSteps = {self.k\_sdl}\nx = {self.x\_sdl}\n')

f\_out.write(f'Check by NumPy:\n')

f\_out.write(f'x = {solve(self.matrix, self.b)}\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

parser.add\_argument('--log', help='Logging')

args = parser.parse\_args()

matrix = Matrix(args.input, args.output, args.log)

matrix.print\_solution()

if \_\_name\_\_ == "\_\_main\_\_":

main()

import argparse

import numpy as np

from numpy.linalg import eig # python .\Lw1\_4.py --input test1 --output out1

def get\_matrix(filename):

with open(filename) as f:

shape = int(f.readline())

matrix = np.array([[float(num) for num in line.split()]

for \_, line in zip(range(shape), f)])

eps = float(f.readline())

return shape, matrix, eps

class Matrix:

def \_\_init\_\_(self, input\_name, output\_name, log\_name):

self.out\_file = output\_name

self.log\_file = log\_name

if log\_name:

open(log\_name, 'w').close()

self.k\_iter = 0

self.shape, self.matrix, self.eps = get\_matrix(input\_name)

self.eigenvalues, self.eigenvectors = self.jacobi\_eigenvalue()

self.eig\_np = eig(self.matrix)

def find\_max(self, matrix):

i\_of\_max = None

j\_of\_max = None

max\_elem = -np.inf

for i in range(self.shape):

for j in range(i + 1, self.shape):

if abs(matrix[i, j]) > max\_elem:

max\_elem = abs(matrix[i, j])

i\_of\_max = i

j\_of\_max = j

if self.log\_file:

with open(self.log\_file, 'a') as fl:

fl.write('Max elem on {0} step: a[{1}][{2}] = {3}\n'.format(

self.k\_iter, i\_of\_max, j\_of\_max, max\_elem))

return i\_of\_max, j\_of\_max

def t(self, matrix):

tmp = np.sqrt(sum([matrix[i, j] \*\* 2 for i in range(self.shape)

for j in range(i + 1, self.shape)]))

if self.log\_file:

with open(self.log\_file, 'a') as fl:

fl.write(f't(A{self.k\_iter}) = {tmp}\n')

return tmp

def jacobi\_eigenvalue(self):

iter\_matrix = self.matrix.copy()

u = np.eye(self.shape)

while True:

u\_matrix = np.eye(self.shape)

i, j = self.find\_max(iter\_matrix)

num1 = iter\_matrix[i, i]

num2 = iter\_matrix[j, j]

if num1 != num2:

phi = np.arctan(2 \* iter\_matrix[i, j] / (num1 - num2)) / 2

else:

phi = np.pi / 4

u\_matrix[i, j] = -np.sin(phi)

u\_matrix[j, i] = np.sin(phi)

u\_matrix[i, i] = np.cos(phi)

u\_matrix[j, j] = np.cos(phi)

u = u @ u\_matrix

iter\_matrix = np.transpose(u\_matrix) @ iter\_matrix @ u\_matrix

self.k\_iter += 1

if self.log\_file:

with open(self.log\_file, 'a') as fl:

fl.write(f'Step {self.k\_iter - 1}:\n')

fl.write(f'Phi = {phi}\n')

fl.write(f'U matrix:\n{u\_matrix}\n')

fl.write(f'Iter matrix:\n{iter\_matrix}\n')

if self.t(iter\_matrix) < self.eps:

break

l = np.diag(iter\_matrix)

return l, u

def print\_solution(self):

if self.log\_file:

with open(self.log\_file, 'a') as f\_log:

f\_log.write(f'General EPS = {self.eps}\nMatrix info:\n')

f\_log.write(f'Shape = {self.shape}\n')

f\_log.write(f'Matrix:\n{self.matrix}\n')

with open(self.out\_file, 'w') as f\_out:

f\_out.write(f'Solution:\nSteps = {self.k\_iter}\n')

f\_out.write(f'Eigenvalues:\n{self.eigenvalues.round(3)}\n')

f\_out.write(f'Eigenvectors:\n{self.eigenvectors.round(3)}\n')

f\_out.write(f'\nCheck by NumPy:\n')

f\_out.write(f'Eigenvalues:\n{self.eig\_np[0].round(3)}\n')

f\_out.write(f'Eigenvectors:\n{self.eig\_np[1].round(3)}\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

parser.add\_argument('--log', help='Logging')

args = parser.parse\_args()

matrix = Matrix(args.input, args.output, args.log)

matrix.print\_solution()

if \_\_name\_\_ == "\_\_main\_\_":

main()

import argparse

import numpy as np

from numpy.linalg import eig, norm

from enum import Enum

def get\_matrix(filename):

with open(filename) as f:

shape = int(f.readline())

matrix = np.array([[float(num) for num in line.split()]

for \_, line in zip(range(shape), f)])

eps = float(f.readline())

return shape, matrix, eps

class Matrix:

class TypeOfEig(Enum):

real = 0

img = 1

def \_\_init\_\_(self, input\_name, output\_name, log\_name):

self.out\_file = output\_name

self.log\_file = log\_name

if log\_name:

open(log\_name, 'w').close()

self.k\_iter = 0

self.shape, self.matrix, self.eps = get\_matrix(input\_name)

self.eigenvalues = self.qr\_algorithm()

self.eig\_np = eig(self.matrix)[0]

def qr\_decomposition(self, r):

q = np.eye(self.shape)

for i in range(self.shape - 1):

h = self.make\_householder(r[:, i], i)

q = q @ h

r = h @ r

if self.log\_file:

with open(self.log\_file, 'a') as f\_log:

f\_log.write(f'=============\n')

f\_log.write(f'For i = {i}\n')

f\_log.write(f'H:\n{h}\n')

f\_log.write(f'Q:\n{h}\n')

f\_log.write(f'R:\n{h}\n')

return q, r

def make\_householder(self, a, k):

v = np.zeros(self.shape)

v[k] = a[k] + np.sign(a[k]) \* norm(a[k:])

for i in range(k + 1, self.shape):

v[i] = a[i]

h = np.eye(self.shape) - (2 / (v @ v)) \* (v[:, None] @ v[None, :])

return h

def qr\_algorithm(self):

a = self.matrix.copy()

flag = True

while True:

q, r = self.qr\_decomposition(a)

a = r @ q # Обратный порядок

is\_eig, type\_eig = zip(\*self.check(a))

if self.log\_file:

with open(self.log\_file, 'a') as f\_log:

f\_log.write(f'=============\n')

f\_log.write(f'{self.k\_iter} th:\n')

f\_log.write(f'Q:\n{q}\n')

f\_log.write(f'R:\n{r}\n')

f\_log.write(f'New A = R \* Q:\n{a}\n')

f\_log.write(f'Is eig:\n{is\_eig}\n')

f\_log.write(f'Type eig:\n{type\_eig}\n')

if all(is\_eig):

if flag:

flag = False

else:

return self.take\_eig(a, type\_eig)

self.k\_iter += 1

def take\_eig(self, matrix, type\_eig):

sol = []

idx = 0

for i in type\_eig:

if i == self.TypeOfEig.real:

sol.append(matrix[idx, idx])

else:

a11 = matrix[idx, idx]

a12 = matrix[idx, idx + 1]

a21 = matrix[idx + 1, idx]

a22 = matrix[idx + 1, idx + 1]

sol.extend(np.roots((1, -a11 - a22, a11 \* a22 - a12 \* a21)))

idx += 1

idx += 1

return np.array(sol)

def check(self, matrix):

check\_eig = []

j = 0

while j < self.shape:

if norm(matrix[j + 1:, j]) <= self.eps:

check\_eig.append((True, self.TypeOfEig.real))

elif norm(matrix[j + 2:, j]) <= self.eps:

check\_eig.append((True, self.TypeOfEig.img))

j += 1

else:

check\_eig.append((False, None))

j += 1

return check\_eig

def print\_solution(self):

if self.log\_file:

with open(self.log\_file, 'a') as f\_log:

f\_log.write(f'=============\n')

f\_log.write(f'General EPS = {self.eps}\nMatrix info:\n')

f\_log.write(f'Shape = {self.shape}\n')

f\_log.write(f'Matrix:\n{self.matrix}\n')

with open(self.out\_file, 'w') as f\_out:

f\_out.write(f'Solution:\nSteps = {self.k\_iter}\n')

f\_out.write(f'Eigenvalues:\n{self.eigenvalues.round(3)}\n')

f\_out.write(f'\nCheck by NumPy:\n')

f\_out.write(f'Eigenvalues:\n{self.eig\_np.round(3)}\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

parser.add\_argument('--log', help='Logging')

args = parser.parse\_args()

matrix = Matrix(args.input, args.output, args.log)

matrix.print\_solution()

if \_\_name\_\_ == "\_\_main\_\_":

main()