Вычислительная математика

Лабораторная работа №1

«Решение системы линейных алгебраических

уравнений СЛАУ»

Вариант №6

Выполнил:

Мухамеджанов Артур Илдусович

P3214

Преподаватель:  
Малышева Татьяна Алексеевна

Код программы:  
<https://github.com/ArturMukhamedjanov/comp_math>

import os.path

def load\_file():

    while(True):

        print("Enter file name:")

        file\_name = input()

        if(os.path.isfile("./" + file\_name)):

            file = open( "./" + file\_name)

            return file

        else:

            continue

def enter\_demension():

    while(True):

        print("type 1 for using file, type 2 for using console input for demension")

        selection = int(input())

        if(selection == 2):

            while(True):

                print("Enter demension:")

                demension = int(input())

                if(demension < 1 or demension > 20):

                    print("Demension should be lower than 20 and higher than 0")

                    continue

                else:

                    return demension

        if(selection == 1):

            while(True):

                file = load\_file()

                line = file.readline()

                demension = int(line.strip())

                if(demension < 1 or demension > 20):

                    print("Demension should be lower than 20 and higher than 0")

                    continue

                else:

                    return demension

def enter\_accuracy():

    while(True):

        print("type 1 for using file, type 2 for using console input for accuracy")

        selection = int(input())

        if(selection == 2):

            while(True):

                print("Enter accuracy:")

                accuracy = float(input())

                if(accuracy < 0):

                    print("Accuracy higher than 0")

                    continue

                else:

                    return accuracy

        if(selection == 1):

            while(True):

                file = load\_file()

                line = file.readline()

                accuracy = float(line.strip())

                if(accuracy < 0):

                    print("Accuracy higher than 0")

                    continue

                else:

                    return accuracy

def enter\_matrix(n):

     while(True):

        print("type 1 for using file, type 2 for using console input for matrix")

        selection = int(input())

        if(selection == 2):

            while(True):

                print("Enter matrix:")

                A = [[0 for \_ in range(n)] for \_ in range(n)]

                B = [0 for \_ in range(n)]

                for i in range(n):

                    numbers = list(map(float, input().split()))

                    if(len(numbers) != n + 1):

                        print("Lines lenght must be n + 1")

                        exit()

                    A[i] = numbers[:n]

                    B[i] = numbers[n]

                return A, B

        if(selection == 1):

            while(True):

                A = [[0 for \_ in range(n)] for \_ in range(n)]

                B = [0 for \_ in range(n)]

                file = load\_file()

                lines = file.readlines()

                if(len(lines) != n):

                    print("Demension of matrix must be n\*n+1 ")

                    continue

                for i in range(n):

                    numbers = lines[i].strip().split(" ")

                    if(len(numbers) != n+1):

                        print("Demension of matrix must be n\*n+1 " + str(numbers))

                        exit()

                    for j in range(n+1):

                        dummy = float(numbers[j])

                        if(j == n):

                            B[i] = dummy

                        else:

                            A[i][j] = dummy

                return A,B

import input

import itertools

def permute\_columns(A, cols, depth, n):

        if depth == n:

            # Создаем новую матрицу с переставленными столбцами

            new\_matrix = [[row[i] for i in cols] for row in A]

            return new\_matrix

        permutations = []

        for i in range(n):

            if i not in cols:

                # Вызываем рекурсивно для следующего столбца

                new\_cols = cols + [i]

                permutations.extend(permute\_columns(A, new\_cols, depth + 1, n))

        return permutations

def check\_matrix\_diag(matrix, n):

    good = True

    strict = False

    for i in range(n):

        sum = 0

        for j in range(n):

            if(i != j):

                sum+=matrix[i][j]

        if sum > matrix[i][i]:

            good = False

            break

        if sum < matrix[i][i]:

            strict = True

    if(strict and good):

        return True

    return False

def decompose\_matrix(matrix, n):

    A = []

    B = []

    for i in range(n):

        A.append(matrix[i][:n])

        B.append(matrix[i][n])

    return A,B

def sort\_diagram(A, B, n):

    full\_matrix = []

    for i in range(n):

        s = A[i]

        s.append(B[i])

        full\_matrix.append(s)

    if check\_matrix\_diag(full\_matrix, n):

        return A, B

    permuntations = itertools.permutations(A)

    for matrix in permuntations:

        if check\_matrix\_diag(matrix, n):

            return decompose\_matrix(matrix, n)

    print("Невозможно составить матрицу c диагональным преобладанием!!! Возможномть несходимость Слау")

    return A, B

def make\_c\_matrix(A, n):

    result\_matrix = []

    for i in range(n):

        dummy = []

        for j in range(n):

            if(i == j):

                dummy.append(0)

            else:

                dummy.append( -A[i][j]/ A[i][i])

        result\_matrix.append(dummy)

    return result\_matrix

def make\_d\_matrix(A, B, n):

    result\_matrix = []

    for i in range(n):

        result\_matrix.append(B[i]/A[i][i])

    return result\_matrix

def sort\_diag\_arr(a, b) :

    n = len(a)

    for i in range(n):

        ind, a[i:] = zip(\*sorted(enumerate(a[i:]),

                             key=lambda x: -x[1][i]))

        b[i:] = [b[i+j] for j in ind]

    return a, b

def iterational\_method(C, D,result\_matrix, accuracy, n, iteration\_counter):

    new\_answers = []

    max\_deviation = 0

    for i in range(n):

        temp = 0

        for j in range(len(C)):

            if(i == j):

                temp += D[j]

            else:

                temp += C[i][j] \* result\_matrix[iteration\_counter][j]

        new\_answers.append(temp)

        if(max\_deviation < abs(result\_matrix[iteration\_counter][i] - temp)):

            max\_deviation = abs(result\_matrix[iteration\_counter][i] - temp)

    new\_answers.append(max\_deviation)

    print(new\_answers)

    result\_matrix.append(new\_answers)

    if(max\_deviation < accuracy):

        return result\_matrix

    else:

        return iterational\_method(C,D, result\_matrix, accuracy, n, iteration\_counter + 1)

n = 3#input.enter\_demension()

accuracy = 0.001#input.enter\_accuracy()

A,B = input.enter\_matrix(n)

#A, B = sort\_diagram(A, B, n)

temp\_A,temp\_B = sort\_diag\_arr(A, B)

if(check\_matrix\_diag(temp\_A, n)):

    A,B = temp\_A, temp\_B

else:

    print("Невозможно составить матрицу c диагональным преобладанием!!! Возможномть несходимость Слау")

C = make\_c\_matrix(A, n)

D = make\_d\_matrix(A, B, n)

temp\_d = D

temp\_d.append(None)

result\_matrix = []

result\_matrix.append(temp\_d)

result\_matrix = iterational\_method(C,D, result\_matrix, accuracy, n, 0)

print(len(result\_matrix))

print(result\_matrix[len(result\_matrix)- 1][:len(result\_matrix[0]) - 1])

Пример работы:  
Входные данные:  
Размерность матрицы: 3  
Точность: 0.01  
Матрица:  
2 2 10 14

10 1 1 12

2 10 1 13

Выходные данные:  
Вектора погрешностей:  
[0.9299999999999998, 0.92, 0.8999999999999999, 0.5]

[1.0179999999999998, 1.024, 1.0299999999999998, 0.1299999999999999]

[0.9945999999999999, 0.9934000000000001, 0.9915999999999999, 0.03839999999999988]

[1.0015, 1.0019200000000001, 1.0024, 0.010800000000000032]

[0.9995679999999999, 0.9994599999999999, 0.9993159999999999, 0.0030840000000000867]

Количество итераций:

6

[0.9995679999999999, 0.9994599999999999, 0.9993159999999999]

Вывод: В ходе данной работы я научился использовать итерационный метод для решения СЛАУ и реализовал его работу на языке Python.