МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №7

Выполнила:
Конаныхина Антонина
P3215
Преподаватель:
Малышева Татьяна
Алексеевна

Санкт-Петербург, 2022

Цель работы

Изучить численные методы решения систем линейных алгебраических уравнений и реализовать один из них средствами программирования.

Задание:

Реализовать метод Гаусса-Зейделя.
Требования:
1) Точность задается с клавиатуры/файла
2) Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.
3) Вывод вектора неизвестных: $x_1,\ x_2, \dots\ x_n$
4) Вывод количества итераций, за которое было найдено решение.
5) Вывод вектора погрешностей: $\left[x_i^{(k)} - x_i^{(k-1)}\right]$

Описание метода

Метод Гаусса-Зейделя является модификацией метода простой итерации и обеспечивает более быструю сходимость к решению системы уравнений. Идея метода: при вычислении компонента $x_i^{(k+1)}$ вектора неизвестных на (k+1)-й итерации используются $x_1^{(k+1)}$, $x_2^{(k+1)}$, … , $x_n^{(k+1)}$, уже вычисленные на (k+1)-й итерации. Значения остальных компонент $x_{i+1}^{(k+1)}$, $x_{i+2}^{(k+1)}$, $x_n^{(k+1)}$ берутся из предыдущей итерации.

Листинг программы

```
import math
import random

INPUT = "input.txt"

def get_matrix_file():
    # Получение матрицы из файла
    with open(INPUT, 'rt') as file:
        try:
            n = int(file.readline())
            acc = float(file.readline())
            max_iterations = int(file.readline())
            matrix = []
            for line in file:
                new_row = list(map(float, line.strip().split()))
                if len(new_row) != (n + 1):
                    raise ValueError
                matrix.append(new_row)
            if len(matrix) != n:
                raise ValueError
        except ValueError:
            return None
    return matrix, n, acc, max_iterations
```

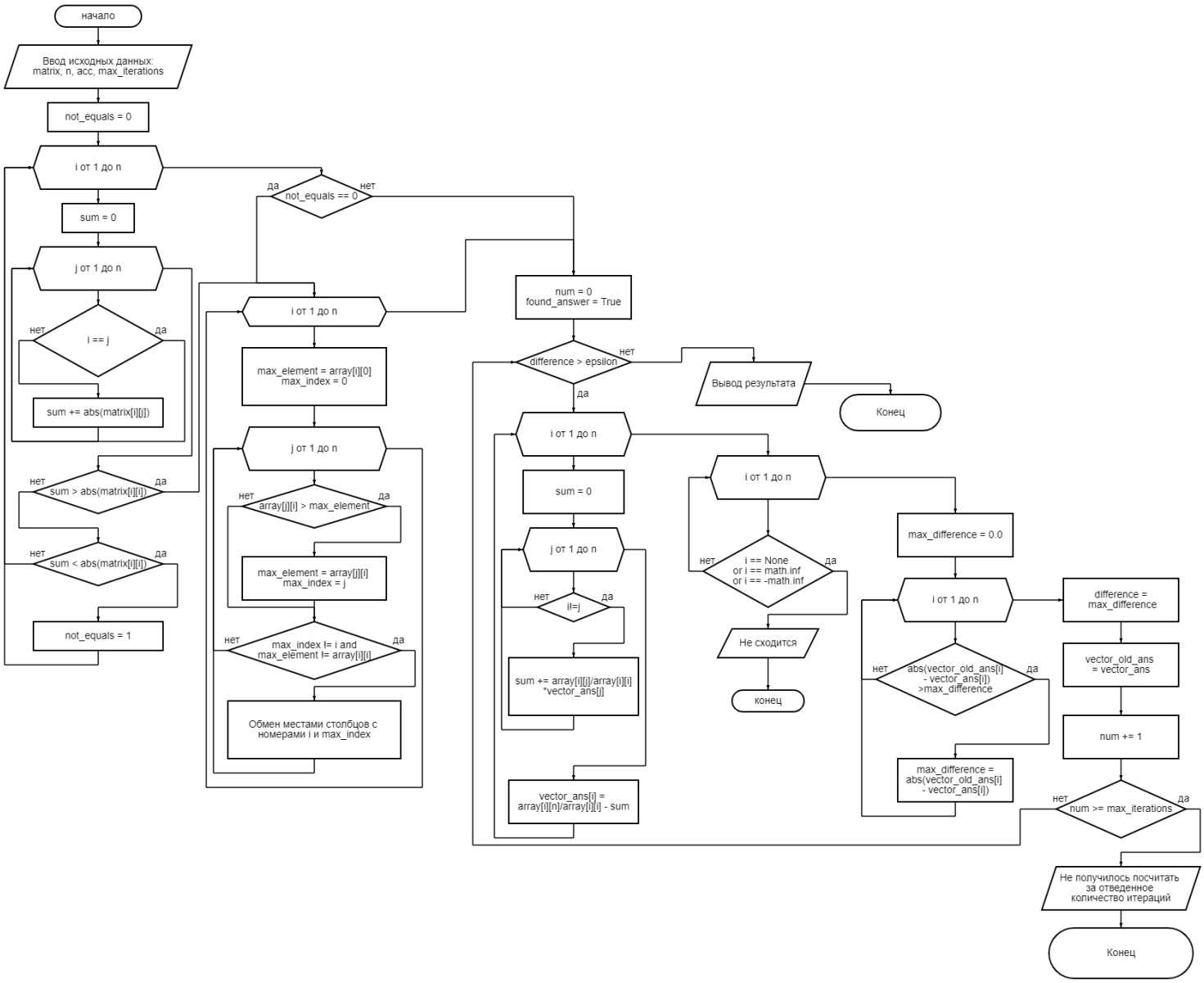
```
def get_matrix_input():
    #Получение матрицы с клавиатуры
    while True:
        try:
            n = int(input("Порядок матрицы: "))
            if n <= 0:
                print("Порядок матрицы должен быть положительным.")
            else:
                break
        except ValueError:
            print("Порядок матрицы должен быть целым числом.")
    while True:
        try:
            acc = float(input("Точность ответа: "))
            if acc <= 0:
                print("Точность матрицы должна быть положительным числом.")
            else:
                break
        except ValueError:
            print("Точность матрицы должна быть числом.")
    while True:
        try:
            max_iterations = int(input("Максимальное количество итераций: "))
            if max_iterations <= 0:
                print("Максимальное количество итераций должно быть
положительным числом.")
            else:
                break
        except ValueError:
            print("Максимальное количество итераций должно быть целым
числом.")
    matrix = []
    print("Введите коэффициенты матрицы через пробел:")
    try:
        for i in range(n):
            matrix.append(list(map(float, input().strip().split())))
            if len(matrix[i]) != (n + 1):
                raise ValueError
    except ValueError:
        return None
    return matrix, n, acc, max_iterations

def get_random_matrix():
    #Создание рандомной матрицы заданного размера
    while True:
        try:
            n = int(input("Порядок матрицы: "))
            if n <= 0:
                print("Порядок матрицы должен быть положительным числом.")
            else:
                break
        except ValueError:
            print("Порядок матрицы должен быть целым числом.")
    while True:
        try:
            acc = float(input("Точность ответа: "))
            if acc <= 0:
                print("Точность матрицы должна быть положительным числом.")
            else:
                break
        except ValueError:
            print("Точность матрицы должна быть числом.")
    while True:
        try:
```

```
            max_iterations = int(input("Максимальное количество итераций: "))
            if max_iterations <= 0:
                print("Максимальное количество итераций должно быть
положительным.")
            else:
                break
        except ValueError:
            print("Максимальное количество итераций должно быть целым
числом.")
    matrix = [[0]*(n+1) for i in range(n)]
    for i in range(n):
        for j in range(n + 1):
            matrix[i][j] = random.randrange(1, 10)
    return matrix, n, acc, max_iterations

#Проверка диагонального преобладания
def check_diagonal(matrix):
    not_equals = 0
    for i in range(len(matrix)):
        sum = 0
        for j in range(len(matrix)):
            if (i == j):
                continue
            sum += abs(matrix[i][j])
        if (sum > abs(matrix[i][i])):
            return False
        if (sum < abs(matrix[i][i])):
            not_equals = 1
            #Если хотя бы для одного строго меньше, то всё классссс
    if not_equals == 0:
        return False
    return True

def print_array(matrix):
    for i in range(len(matrix)):
        output = ""
        for j in range(len(matrix)):
            output += str(matrix[i][j]) + " "
        output += "| " + str(matrix[i][len(matrix)])
        print(output)

# main logic

input_format = ""

while input_format != "1" and input_format != "2" and input_format != "3":
    input_format = input("Введите \"1\" - чтобы ввести матрицу с клавиатуры,
\"2\" - чтобы взять матрицу из файла, \"3\" - заполнить случайными числами:
")

if input_format == "1":
    array, n, epsilon, max_iterations = get_matrix_input()
elif input_format == "2":
    array, n, epsilon, max_iterations = get_matrix_file()
else:
    array, n, epsilon, max_iterations = get_random_matrix()

print("Полученный массив:")

print_array(array)

print("Полученная точность: " + str(epsilon))

print("Полученное количество итераций: " + str(max_iterations))
```

```
#Сохранение порядка неизвестных первоначальной матрицы
answer_order = [i for i in range(n)]

if (check_diagonal(array)):
    print("Выполнено условие преобладания диагональных элементов")
else:
    print("Не выполнено условие преобладания диагональных элементов")
    for i in range(n):
        max_element = array[i][0]
        max_index = 0
        for j in range(n):
            if (array[j][i] > max_element):
                max_element = array[j][i]
                max_index = j
        if (max_index != i and max_element != array[i][i]):
            #Меняем местами столбцы, если максимальный элемент не на
диагонали
            for j in range(n):
                array[j][i], array[j][max_index] = array[j][max_index],
array[j][i]
        answer_order[i], answer_order[max_index] = answer_order[max_index],
answer_order[i]

        if(check_diagonal(array)):
            #проверим, не выполняется ли теперь условие диагонального
преобладания. если да, то всё супер
            break
    if (check_diagonal(array)):
        print("Матрицу получилось привести к виду, в котором выполняется
условие преобладания диагонали")
    else:
        print("Матрицу не получилось привести к виду, в котором выполняется
условие преобладания диагонали")
    print("Новый вид матрицы:")
    print_array(array)

for i in range(n):
    if (array[i][i] == 0):
        #Если на диагонали вдруг оказался ноль, значит будет деление на ноль,
а это очень плохо :(
        print("Не получится найти ответ, так как на диагонали есть 0")
        exit()

vector_old_ans = [0]*n
vector_ans = [0]*n
difference = epsilon + 1
num = 0
found_answer = True
errors = [0]*n
while difference > epsilon:
    for i in range (n):
        sum = 0
        for j in range (n):
            if i!=j:
              sum += array[i][j]/array[i][i]*vector_ans[j]
        vector_ans[i] = array[i][n]/array[i][i] - sum
    for i in vector_ans:
        if i == None or i == math.inf or i == -math.inf:
            print("Значения расходятся, невозможно найти ответ")
            found_answer = False
            exit(0)

    max_difference = 0.0
```

```
    for i in range(n):
        errors[i] = abs(vector_old_ans[answer_order[i]] -
vector_ans[answer_order[i]])
        if abs(vector_old_ans[i] - vector_ans[i])>max_difference:
            max_difference = abs(vector_old_ans[i] - vector_ans[i])
    difference = max_difference
    for i in range(n):
        vector_old_ans[i] = vector_ans[i]
    num += 1
    if (num >= max_iterations):
        print("Не удалось получить ответ за максимальное количество
итераций")
        found_answer = False
        break
if (found_answer):
    print("Ответ найден за " + str(num) + " итераций:")
    # print(vector_ans)
    print([vector_ans[answer_order[i]] for i in range(n)])
    print("Вектор погрешностей:")
    print(errors)
```

Блок-схема метода



Примеры работы программы

```
Введите "1" - чтобы ввести матрицу с клавиатуры, "2" - чтобы взять матрицу из файла, "3" - заполнить случайными числами: 2
Полученный массив:
8.0 2.0 2.0 3.0 | 8.0
2.0 15.0 2.0 4.0 | 2.0
1.0 2.0 2.0 22.0 | 2.0
1.0 3.0 23.0 2.0 | 4.0
Полученная точность: 0.001
Полученное количество итераций: 20
Не выполнено условие преобладания диагональных элементов
Матрицу получилось привести к виду, в котором выполняется условие преобладания диагонали
Новый вид матрицы:
8.0 2.0 3.0 2.0 | 8.0
2.0 15.0 4.0 2.0 | 2.0
1.0 2.0 22.0 2.0 | 2.0
1.0 3.0 2.0 23.0 | 4.0
Ответ найден за 4 итераций:
[0.9585758349344986, -0.02202640305311304, 0.13186112929484767, 0.037348700221671874]
Вектор погрешностей:
[0.00032222264950743096, 0.00021927344463637533, 4.165385262541732e-05, 1.1002186516145818e-05]

Process finished with exit code 0
```

Вывод:

В результате выполнения данной лабораторной работой я познакомилась с численными методами решения математических задач на примере систем алгебраических уравнений, реализовав на языке программирования Python метод Гаусса-Зейделя.