Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»
Факультет Программной Инженерии и Компьютерной Техники

Вариант №9
Лабораторная работа №1
по теме
Решение системы линейных алгебраических уравнений СЛАУ
по дисциплине
Вычислительная математика

Выполнил Студент группы P3212
Кобелев Р.П.
к. т. н. Преподаватель:
Наумова Н.А.

г. Санкт-Петербург
2024г.

Содержание

1 Цель работы

Разработать приложение для решения систем линейных алгебраических уравнений методом Гаусса-Зейгеля.

2 Задание

1. В программе численный метод должен быть реализован в виде отдельной подпрограммы/метода/класса, в который исходные/выходные данные передаются в качестве параметров.

2. Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя).

3. Точность задается с клавиатуры/файла

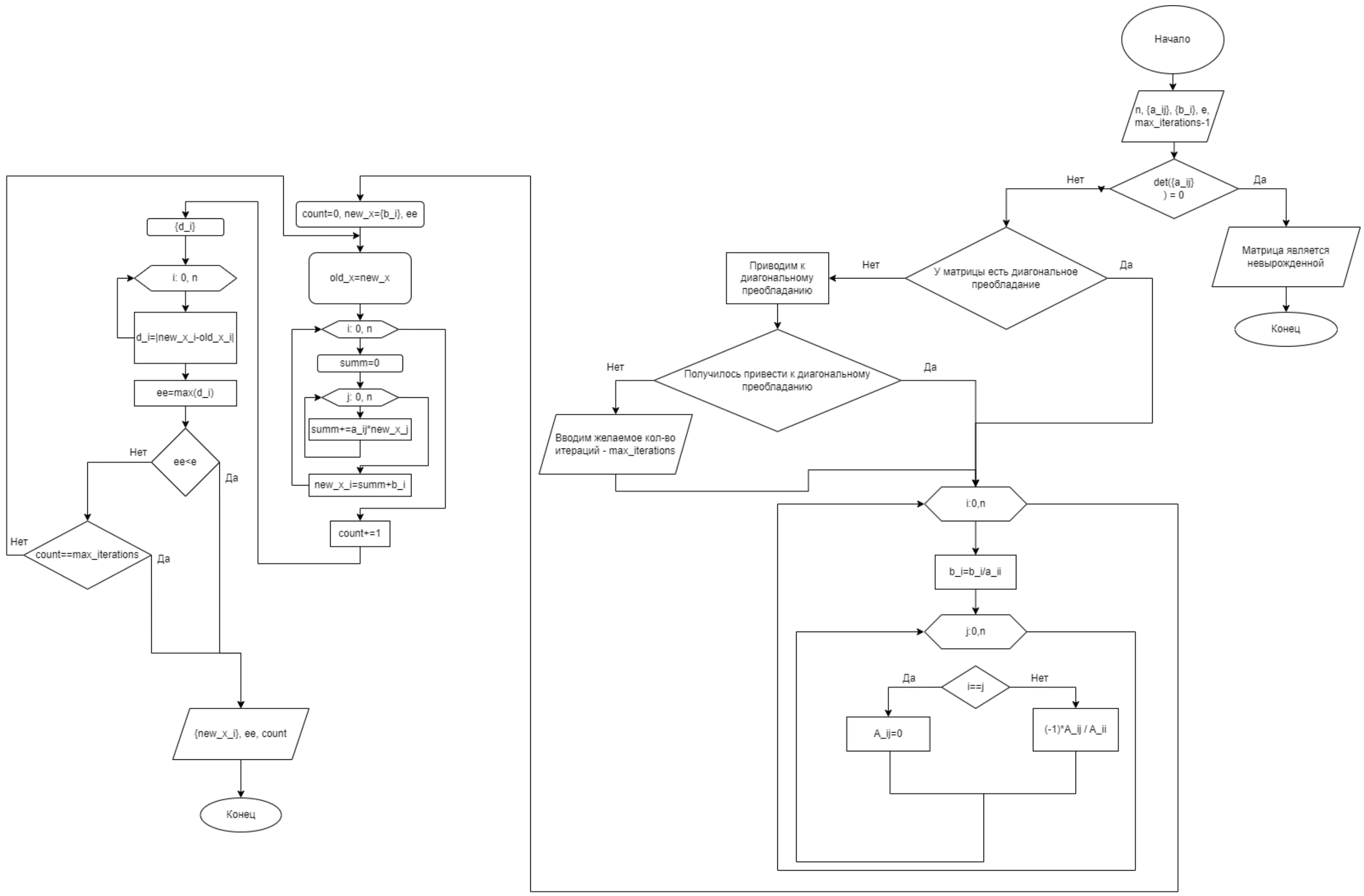
4. Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.

5. Вывод вектора неизвестных: $x_1, x_2, X_3...$

6. Вывод количества итераций, за которое было найдено решение

7. Вывод вектора погрешностей: $\left|x_i^k - x_i^{k-1}\right|$

3 Блок-схема реализованного алгоритма



4 Реализация (код) численного метода

Solver.py

```
class GaussSeidelSolver:
    def __init__(self, A, b, accuracy):
        self.A = A
        self.b = b
        self.accuracy = accuracy
        self.max_iterations = -1
        if not self.is_matrix_non_singular():
            raise ValueError(”Матрица вырожденная. Решениеневозможно.”)

    # Проверканевырожденностиматрицы
    def is_matrix_non_singular(self):
        return self.calculate_determinant(self.A) != 0

    def make_diagonally_dominant(self, matrix):
        n = len(matrix)

        for i in range(n):
            diagonal_element = abs(float(matrix[i][i]))
            row_sum = sum(map(abs, map(float, matrix[i]))) - diagonal_element

            if diagonal_element > row_sum:
                continue

            max_element = max(map(abs, map(float, matrix[i])))
            try:
                max_element_index = matrix[i].index(max_element)
            except ValueError:
                max_element_index = matrix[i].index(-max_element)

            matrix[i], matrix[max_element_index] = matrix[max_element_index], matrix[i]
            self.b[i], self.b[max_element_index] = self.b[max_element_index], self.b[i]

        return matrix

    # Рекурсивныйметодвычисленияопределителя
    def calculate_determinant(self, matrix):
        det = 1.0
        n = len(matrix)

        for i in range(n):
            max_row = max(range(i, n), key=lambda j: abs(matrix[j][i]))
            if matrix[max_row][i] == 0.0:
                return 0.0
            if max_row != i:
                matrix[i], matrix[max_row] = matrix[max_row], matrix[i]
                det *= -1.0

            pivot = matrix[i][i]
            det *= pivot

            for j in range(i + 1, n):
```

```
                factor = matrix[j][i] / pivot
                for k in range(i, n):
                    matrix[j][k] -= factor * matrix[i][k]
        print(f”Значение детерминанта: {det}”)
        return det

    # Проверкадиагональногопреобладания
    def is_diagonally_dominant(self, matrix):
        n = len(matrix)

        for i in range(n):
            diagonal_element = float(matrix[i][i])
            row_sum = sum(map(float, matrix[i])) - diagonal_element

            if diagonal_element <= row_sum:
                return False

        return True

    # Приведемсистему□A = □квиду□= □□+ □
    def transform(self):
        self.b = [bi / self.A[i][i] for i, bi in enumerate(self.b)]
        self.A = [
            [0.0 if i == j else (-1) * d / self.A[i][i] for j, d in enumerate(row)]
            for i, row in enumerate(self.A)
        ]

    def get_accuracy(self, new_x, old_x):
        return max(abs(a - b) for a, b in zip(new_x, old_x))

    def is_accuracy_reached(self, new_x, old_x, precision):
        return max(abs(a - b) for a, b in zip(new_x, old_x)) < precision

    def iterate(self):
        count = 0
        new_x = list(self.b)
        while True:
            old_x = list(new_x)
            for i in range(len(new_x)):
                new_x[i] = (
                    sum(self.A[i][j] * d for j, d in enumerate(new_x)) + self.b[i]
                )
            count += 1
            if (
                self.is_accuracy_reached(new_x, old_x, self.accuracy)
                or count == self.max_iterations
            ):
                break
        return [[new_x], self.get_accuracy(new_x, old_x), count]

    def solve(self):
        if not self.is_diagonally_dominant(self.A):
            print(”Изначальная матрицанеимеетдиагональногопреобладания”)
            old_A= list(self.A)
            self.A = self.make_diagonally_dominant(self.A)
            if self.is_diagonally_dominant(self.A):
```

```
                print("Получилось привестиматрицукдиагональномупреобладанию")
            else:
                print("Матрицу неполучилосьпривестикдиагональномупреобладанию\n")
                self.A=list(old_A)
                while True:
                    try:
                        self.max_iterations = int(input("Введите желаемоеколичествоитераций: "))
                        break
                    except ValueError:
                        print("Ошибка: Введенонекорректноечисло. Пожалуйста, введитецелоечисло.")


        self.transform()
        return self.iterate()
```

5 Ссылка на GitHub с основной реализацией

Ссылка на GitHub

6 Пример работы программы

Полученная матрица:
4.826 89.63062 82.80024 69.55934 35.65224
7.98158 27.34614 37.06014 35.80124 92.69239
85.41202 12.39656 62.29112 67.52862 66.42124
57.91106 65.25669 91.87075 19.12514 35.0888
50.54602 41.88476 81.1894 79.40341 16.2745
Полученный вектор ответов:
63.29296 2.58799 50.77971 79.99869 36.93007
Заданная погрешность: 0.001

Значение детерминанта: 672277389.867646
Изначальная матрица не имеет диагонального преобладания
Матрицу не получилось привести к диагональному преобладанию

Введите желаемое количество итераций: 4
Получившийся вектор:
, -4.072850842724372, 5.5841225936857635, -1.4398844916097835, 0.5247165140858333

Получившаяся погрешность: 0.03413357461361066
Количество итераций: 4

7 Вывод

В ходе реализации данной лабораторной работы я ознакомился с работой алгоритма Гаусса-Зейгеля, предназначенного для решения совместных определенных систем линейных алгебраических уравнений (СЛАУ). Данный алгоритм относится к виду итерационных: решение системы (если оно существует) достигается путем приближения за счет конечного числа итераций.