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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №1
«Решение СЛАУ»
Вариант №2

Группа P3208

Студент Горин Семён Дмитриевич

Преподаватель Рыбаков Степан Дмитриевич

Санкт-Петербург, 2026

Содержание

1. Цель работы

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

2. Описание метода

Метод Гаусса.

Основан на приведении матрицы системы к треугольному виду так, чтобы ниже ее главной диагонали находились только нулевые элементы. Состоит из 2 фаз: прямого и обратного хода.

Прямой ход метода Гаусса состоит в последовательном исключении неизвестных из уравнений системы. Сначала с помощью первого уравнения исключается x_1 из всех последующих уравнений системы. Затем с помощью второго уравнения исключается x_2 из третьего и всех последующих уравнений и т.д. Этот процесс продолжается до тех пор, пока в левой части последнего (n-го) уравнения не останется лишь один член с неизвестным x_n, т. е. матрица системы будет приведена к треугольному виду.

Обратный ход метода Гаусса состоит в последовательном вычислении искомых неизвестных: решая последнее уравнение, находим неизвестное x_n. Далее, из предыдущего уравнения вычисляем x_{n-1} и т. д. Последним найдем x_1 из первого уравнения.

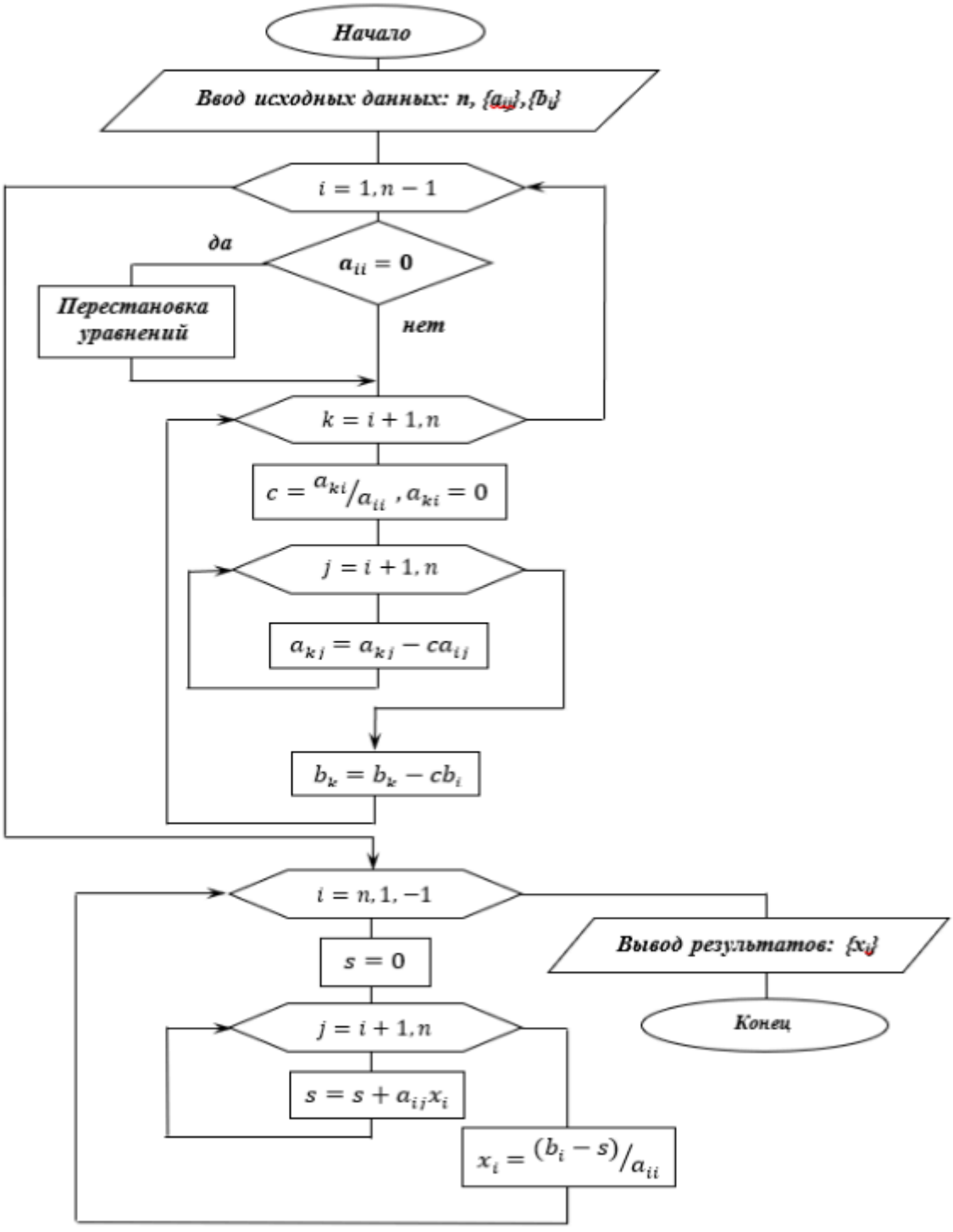

Рис. 1 — Блок-схема метода Гаусса

3. Листинг

Далее приведен листинг функций используемых для нахождения определителя, треугольного представления матрицы коэффициентов, вектора решений и вектора невязок. Также приведены функции с использованием сторонних библиотек для нахождения всего вышеперечисленного.

```
1:
2:
3: def find_determinant(matrix: list[list[float]]) -> float:
4:     """Определитель матрицы"""
5:     n = len(matrix)
```

```
6:
7:     triangle_matrix, swap_count = find_triangle_matrix(matrix)
8:
9:     det = 1.0
10:    for i in range(n):
11:        det *= triangle_matrix[i][i]
12:
13:    if swap_count % 2 != 0:
14:        det = -det
15:
16:    return det
17:
18:
19: def find_triangle_matrix(matrix: list[list[float]]) ->
    tuple[list[list[float]], int]:
20:    """Приведение квадратной матрицы к треугольному виду."""
21:    matrix = [row[:] for row in matrix] # Копируем матрицу
22:    n = len(matrix)
23:    swap_count = 0
24:    for i in range(n):
25:
26:        pivot_row = None
27:        for j in range(i, n): # проходимся по строке и ищем по какому
    из элементов можно приводить матрицу
28:            if (abs(matrix[j][i]) > 1e-12):
29:                pivot_row = j
30:                break
31:
32:        if (pivot_row != None):
33:            if (pivot_row != i): # если строка стоит не на своем месте
    меняем местами
34:                matrix[pivot_row], matrix[i] = matrix[i],
    matrix[pivot_row]
35:                swap_count += 1
36:
37:            for j in range(i + 1, n): # проходимся по всем строкам
    ниже и вычитаем из каждой строку по которой приводим
38:                factor = matrix[j][i] / matrix[i][i] # множитель для
    конкретной строки
39:                for k in range(i, len(matrix[j])):
40:                    matrix[j][k] -= factor * matrix[i][k]
41:        else:
42:            raise ValueError("Матрицу невозможно привести к
    треугольному виду.")
43:    return matrix, swap_count
44:
45:
46: def find_variable_vector(matrix: list[list[float]]) -> list[float]:
47:    """Нахождение вектора решений"""
48:    n = len(matrix)
49:
50:    matrix = [row[:] for row in matrix] # Копируем матрицу
51:    matrix, _ = find_triangle_matrix(matrix)
```

```
52:     ans = []
53:
54:     for i in range(n - 1, -1, -1):
55:         matrix[i][n] /= matrix[i][i]
56:         for j in range(i - 1, -1, -1):
57:             matrix[j][n] -= matrix[j][i] * matrix[i][n]
58:         ans.append(matrix[i][n])
59:     return ans[::-1]
60:
61: def find_vector_of_residuals(matrix: list[list[float]]) ->
    list[float]:
62:     """Нахождение вектора невязок"""
63:     n = len(matrix)
64:     matrix = [row[:] for row in matrix] # Копируем матрицу
65:     variables = find_variable_vector(matrix)
66:     residuals = [0] * n
67:     for i in range(n):
68:         for j in range(n):
69:                 residuals[i] += matrix[i][j] * variables[j]
70:         residuals[i] -= matrix[i][n]
71:     return residuals
72:
73: import numpy as np
74:
75: def np_find_determinant(matrix: list[list[float]]) -> float:
76:     A = np.array(matrix, dtype=float)
77:     A = A[:, : -1]
78:     return float(np.linalg.det(A))
79:
80: def np_find_triangle_matrix(matrix: list[list[float]]) ->
    list[list[float]]:
81:     A = np.array(matrix, dtype=float)
82:
83:     Q, R = np.linalg.qr(A)
84:
85:     return R.tolist()
86:
87: def np_find_variable_vector(matrix: list[list[float]]) -> list[float]:
88:     A = np.array([row[:-1] for row in matrix], dtype=float)
89:     b = np.array([row[-1] for row in matrix], dtype=float)
90:
91:     x = np.linalg.solve(A, b)
92:     return x.tolist()
93:
94: def np_find_vector_of_residuals(matrix: list[list[float]]) ->
    list[float]:
95:     A = np.array([row[:-1] for row in matrix], dtype=float)
96:     b = np.array([row[-1] for row in matrix], dtype=float)
97:
98:     x = np.linalg.solve(A, b)
99:
100:     residuals = A @ x - b
101:     return residuals.tolist()
```

4. Примеры работы программы

На следующем листинге представлен пример работы программы с вводом данных из консоли и подсчетом всех параметров при помощи самописных функций:

```
> all_info
Введите матрицу построчно, каждый новый элемент должен быть отделен от предыдущего пробелом. Для выхода из режима ввода нажмите esc, а затем enter.
Пример ввода:
1 2 3
4 5 6
7 8 9

1 1
Определитель:
1.0
Преобразованнная матрица:
+-----+-----+
| 1.0 | 1.0 |
+-----+-----+
Решение методом Гаусса:
x[1] = 1.0
Вектор невязки:
r[1] = 0.0
```

На следующем листинге представлен пример работы программы с вводом данных из файла и подсчетом всех параметров при помощи библиотеки «numpy»:

```
> all_info --from-file test.txt --numpy
Определитель:
1.0
Преобразованнная матрица:
+-----+-----+
| 1.0 | 1.0 |
+-----+-----+
Решение методом Гаусса:
x[1] = 1.0
Вектор невязки:
r[1] = 0.0
```

5. Вывод

В результате выполнения лабораторной работы я познакомился с численными методами решения СЛАУ, и реализовал метод Гаусса на языке программирования Python.

Сравнивая результат работы программы написанной мной и библиотеки «numpy» можно заметить, что при достаточно несбалансированных матрицах результат может отличаться. Это происходит из-за того что при вычислении необходимых параметров библиотекой «numpy» используется другой числен-

ный метод, из-за чего результат операций с числами с плавающей точкой может незначительно отличаться.